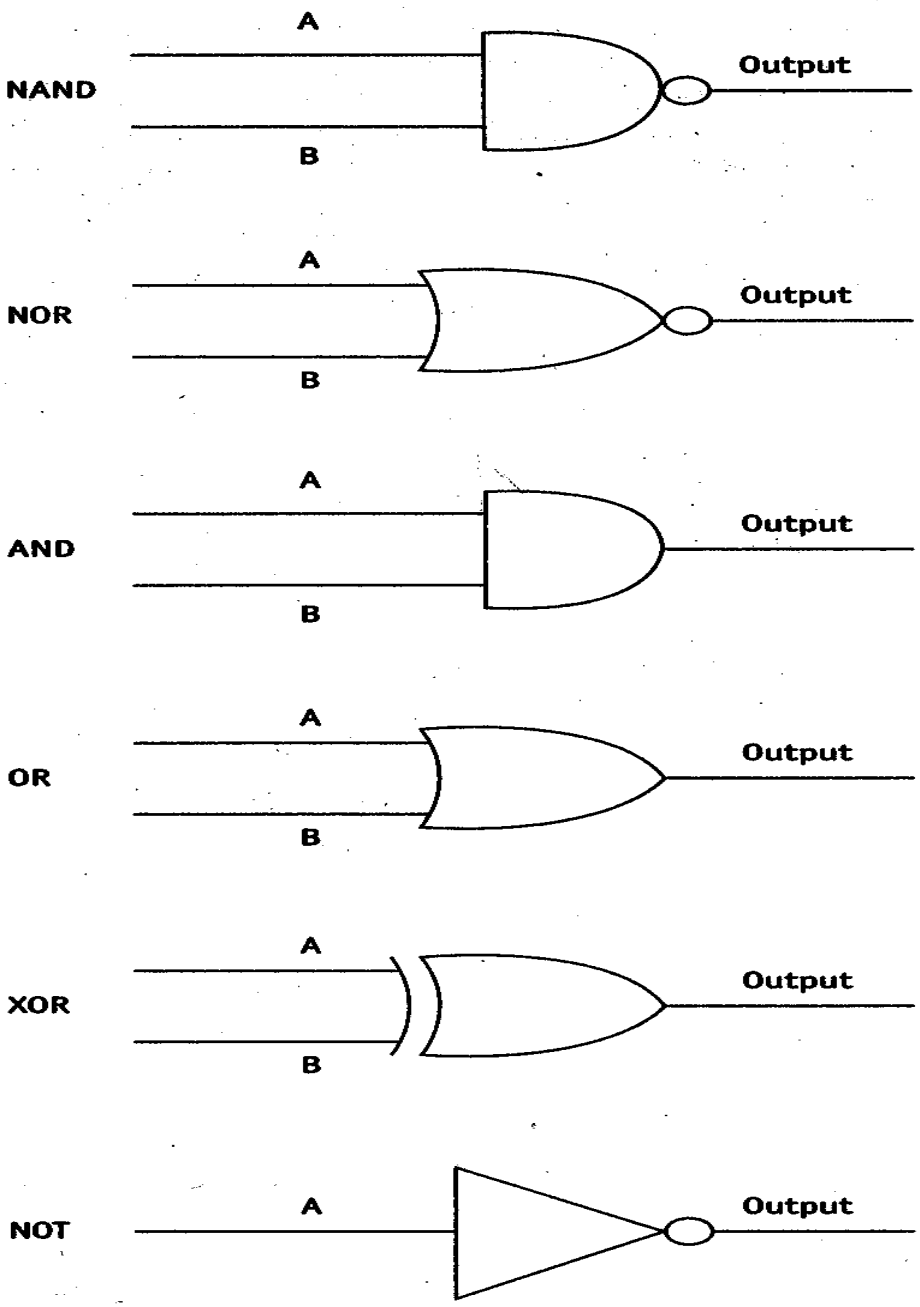
Chapter 6

Microlevel of H1 and V1

We start with some concepts from Chapter 5 that are essential for this chapter.

FIGURE 5.9



A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

A	Output
0	1
1	0

FIGURE 5.11 Gating Action

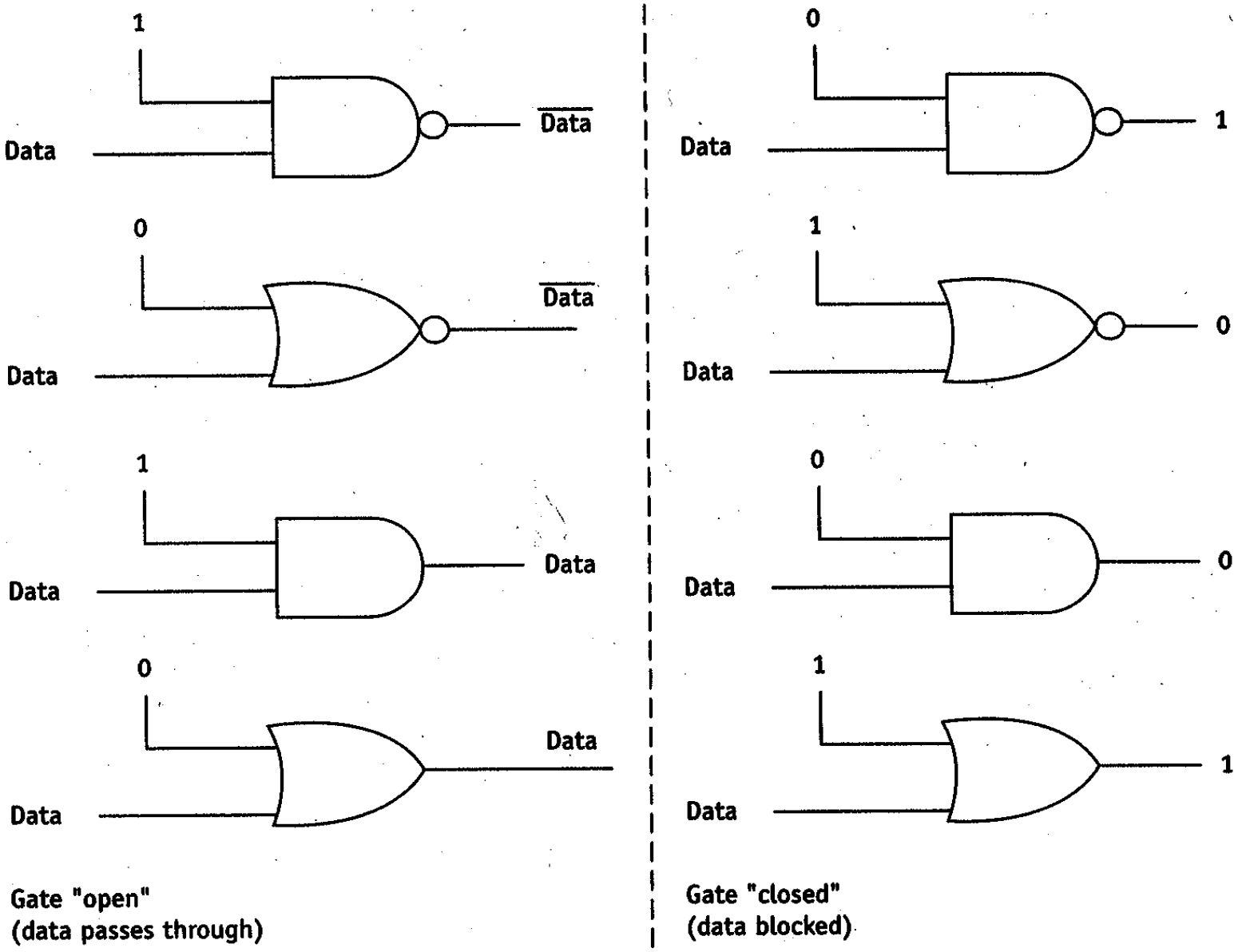


FIGURE 5.22 4-Input Multiplexer

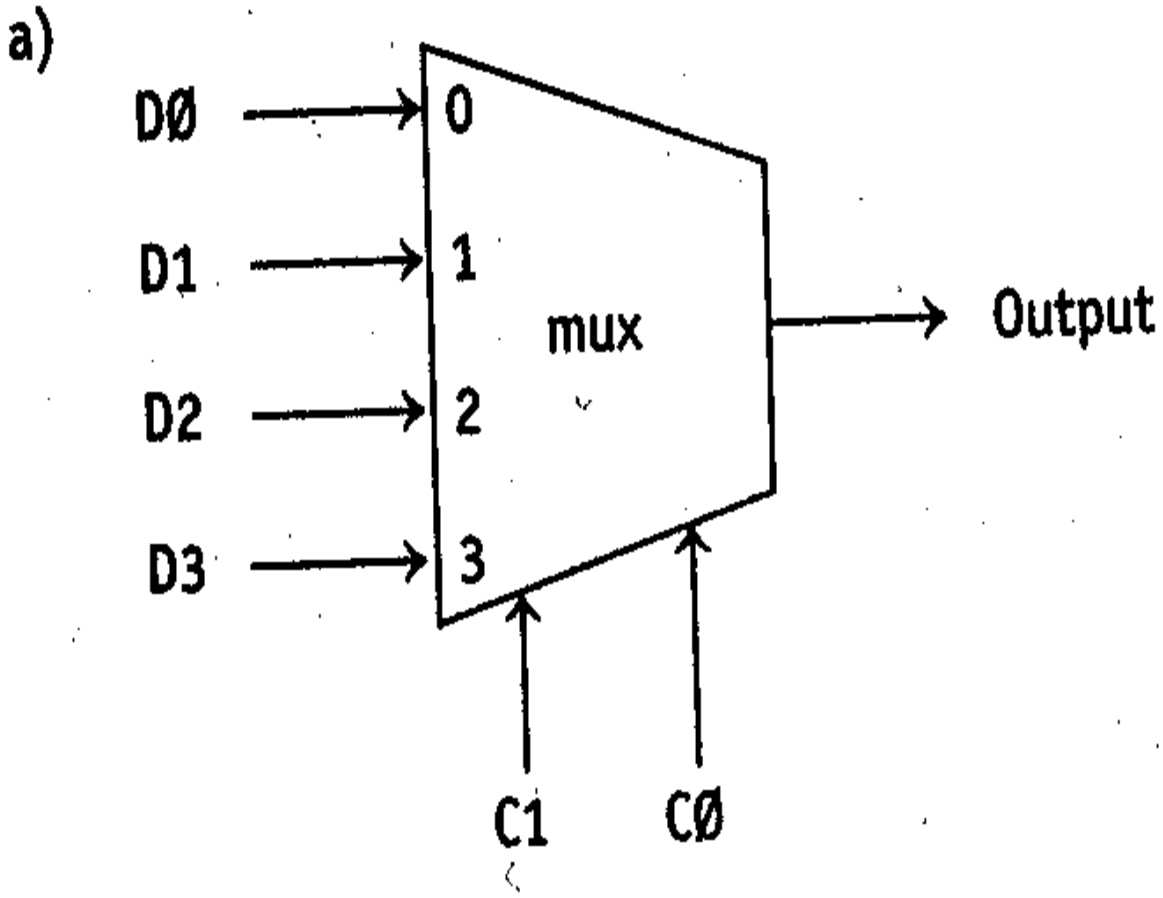


FIGURE 5.23

Two-Bus Input Multiplexer

a)

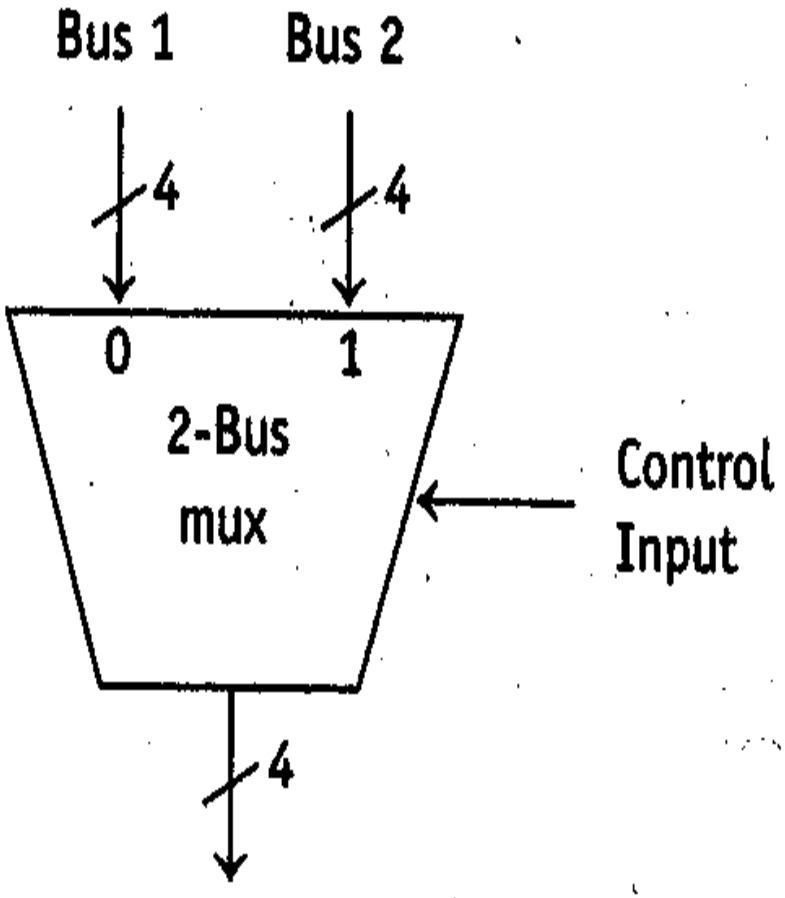


FIGURE 5.24 Two-Input Decoder

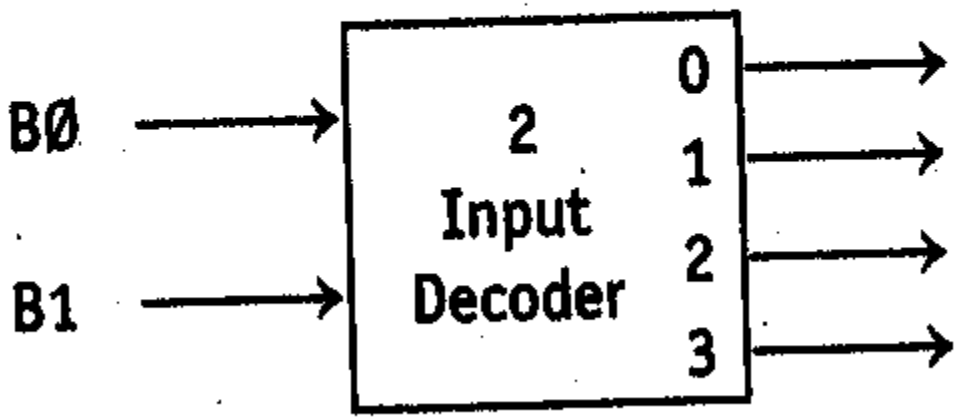
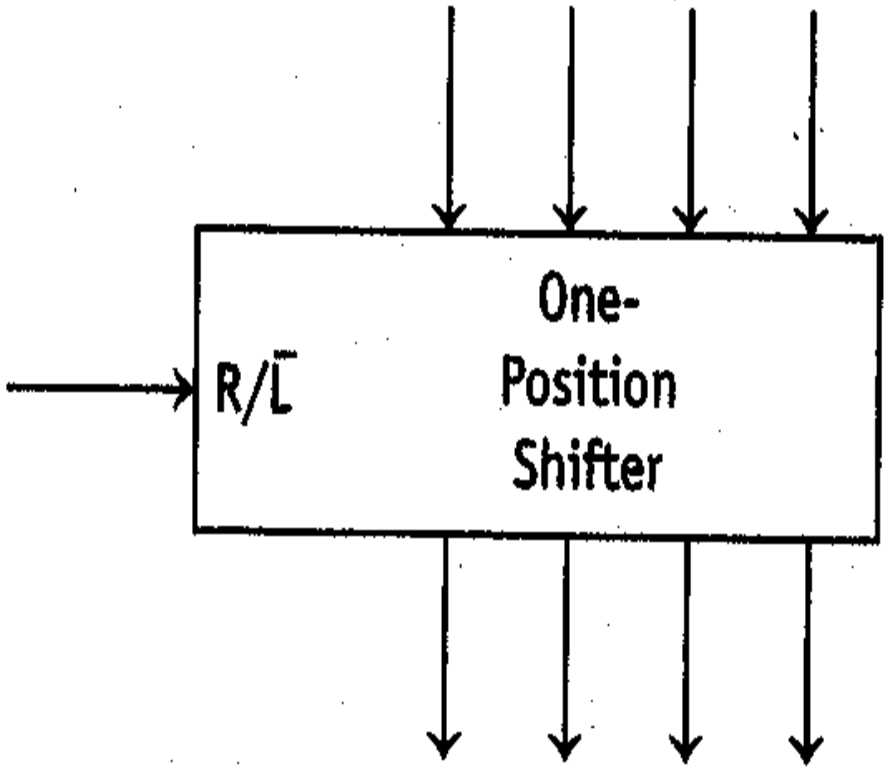


FIGURE 5.27

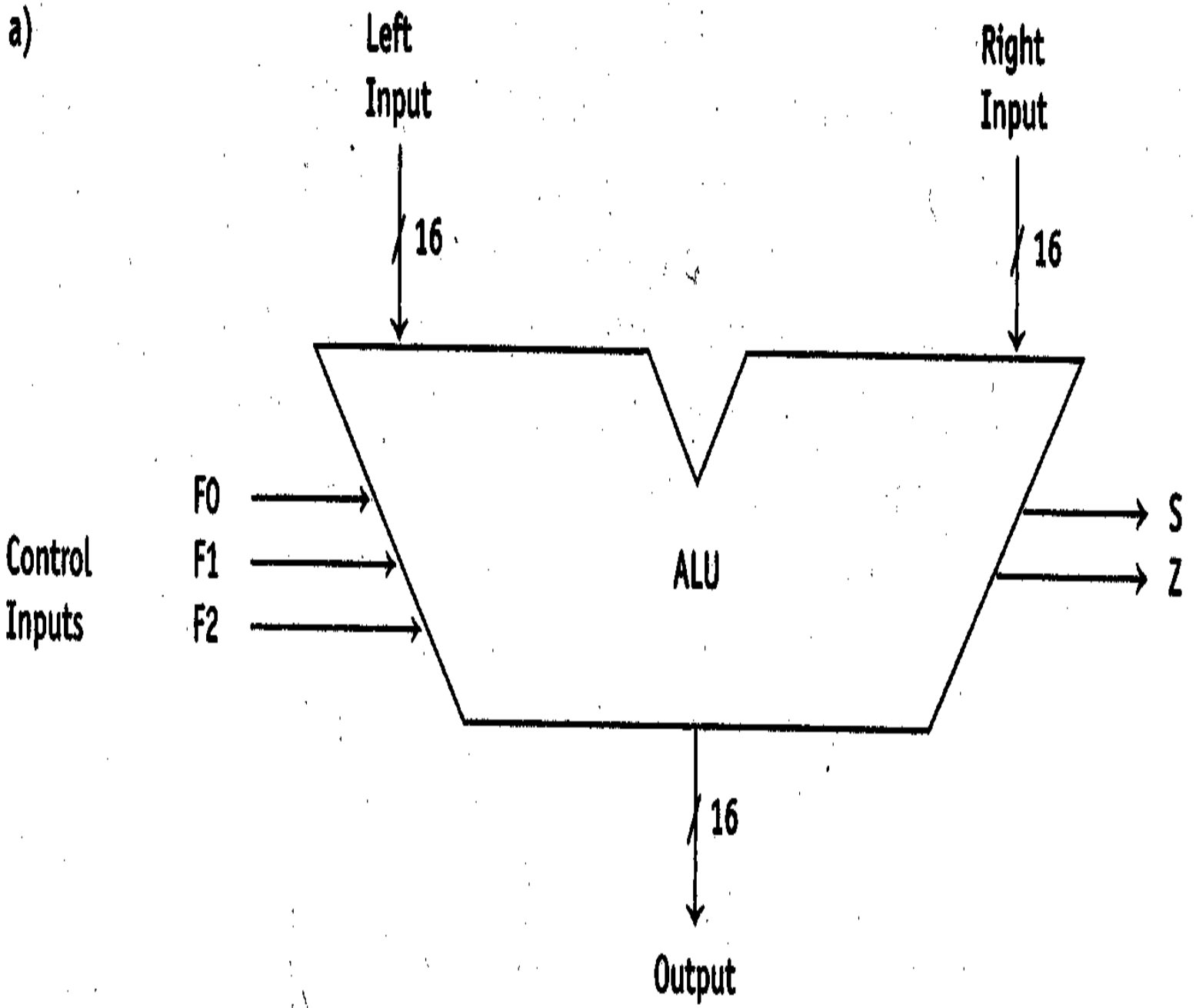
One-Position Shifter

a)



b)

FIGURE 5.29


a)
Left
Input
16
Right
Input
16
F0
F1
F2
Control
Inputs
ALU
S
Z
16
Output

ALU functions

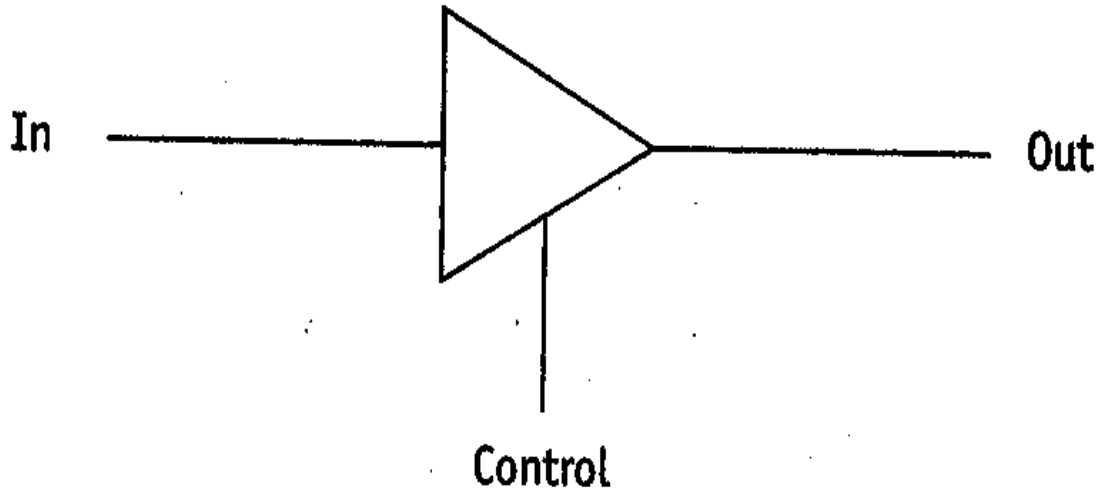
b)

F2	F1	F0	Output	
0	0	0	`left`	output is same as left input
0	0	1	`~left`	bitwise complement left input
0	1	0	`left & right`	AND inputs
0	1	1	`left * right`	multiply inputs
1	0	0	`left + right`	add inputs
1	0	1	`left - right`	subtract inputs
1	1	0	`left << 1`	left shift left input one position
1	1	1	`left >> 1`	right shift left input one position

FIGURE 5.33

Tri-State Buffer

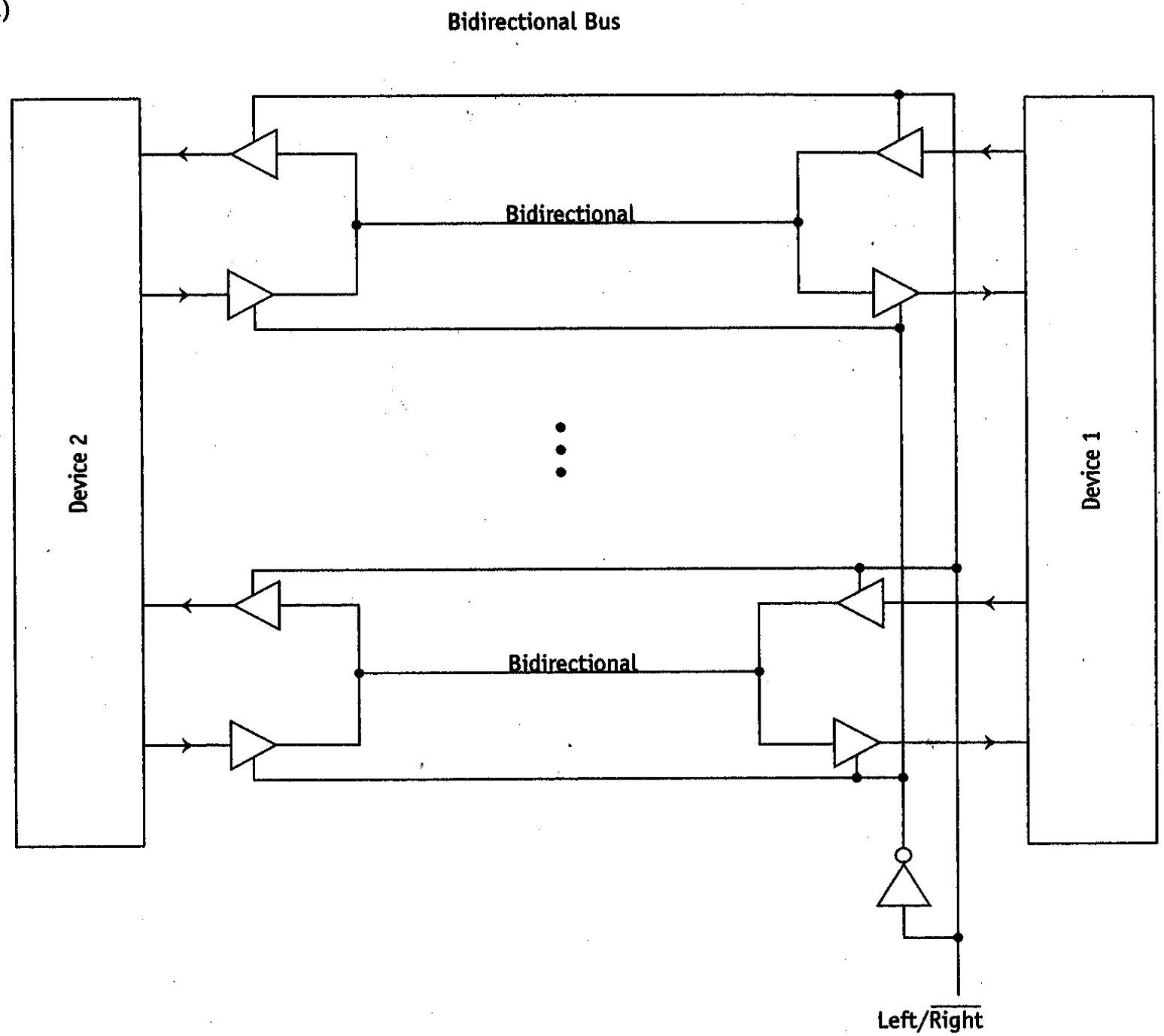
a)



b)

Control	In	Out
0	0	High Z
0	1	High Z
1	0	0
1	1	1

FIGURE 5.34 a)



b)

FIGURE 5.47

Read/Write Register
(Synchronous)

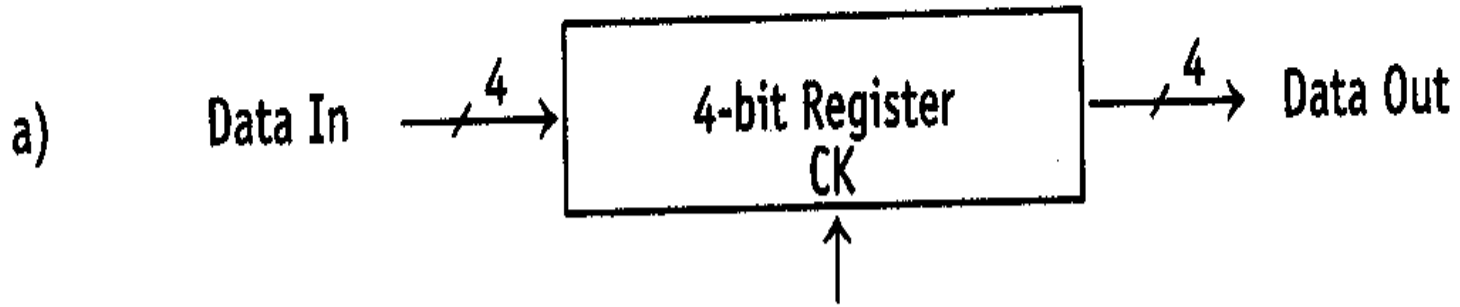
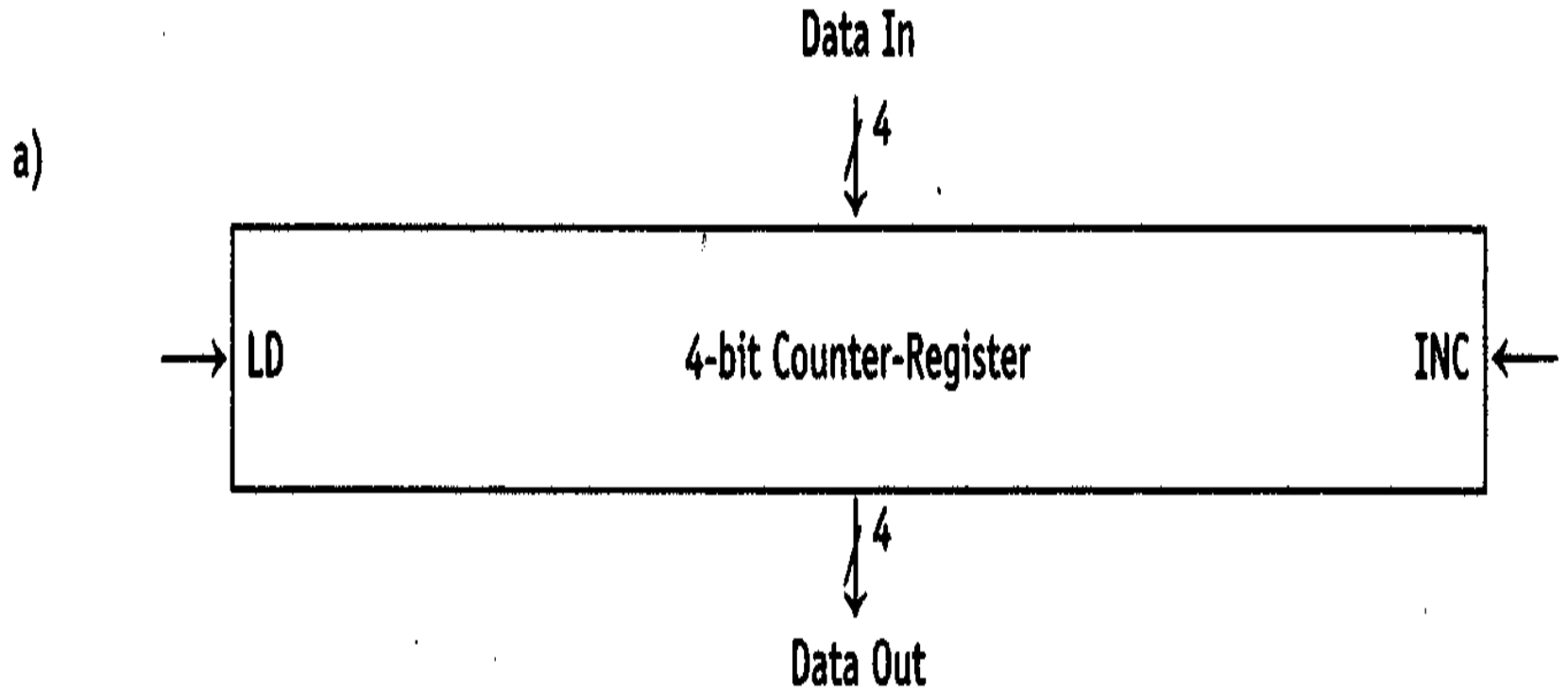


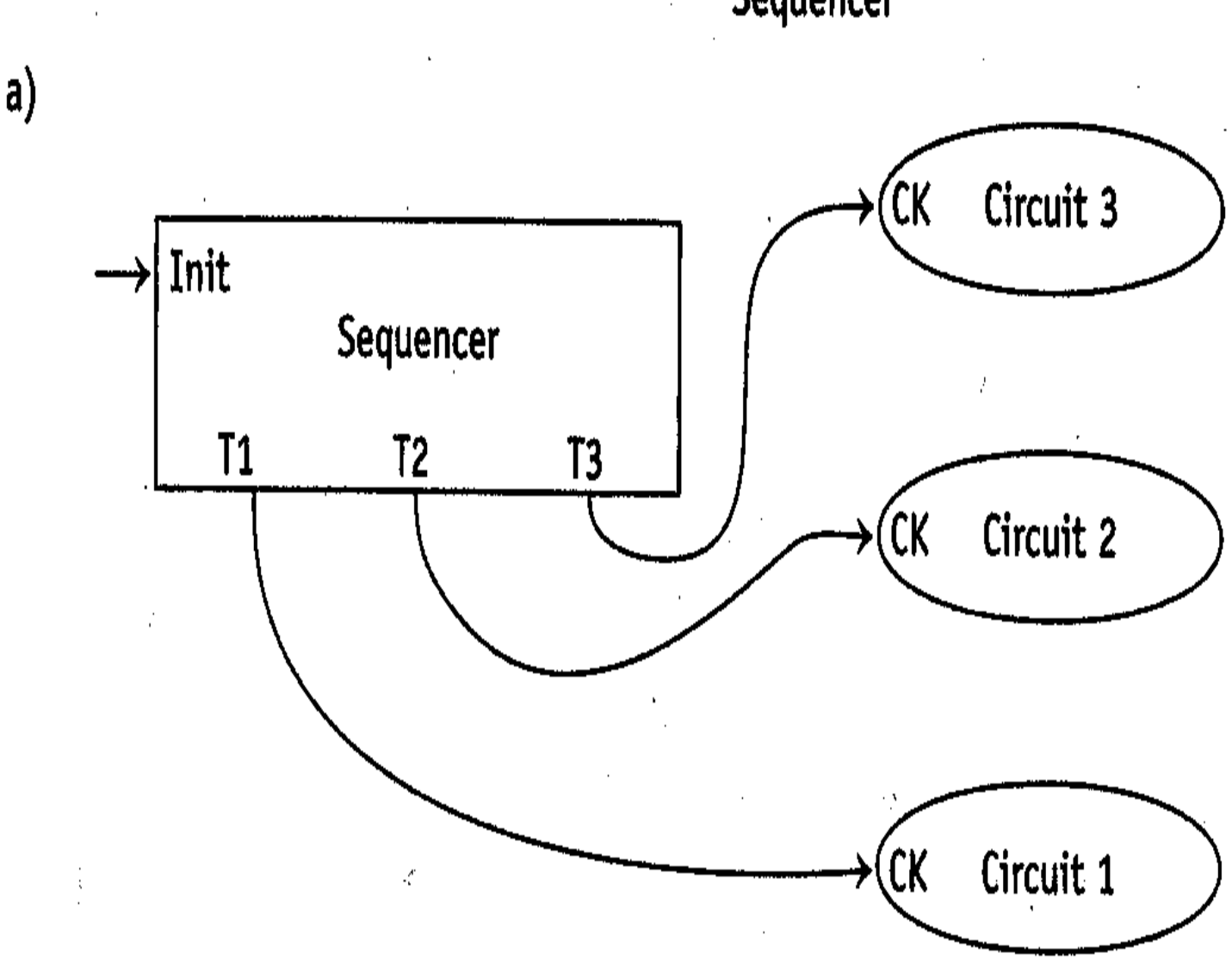
FIGURE 5.48 Counter-Register



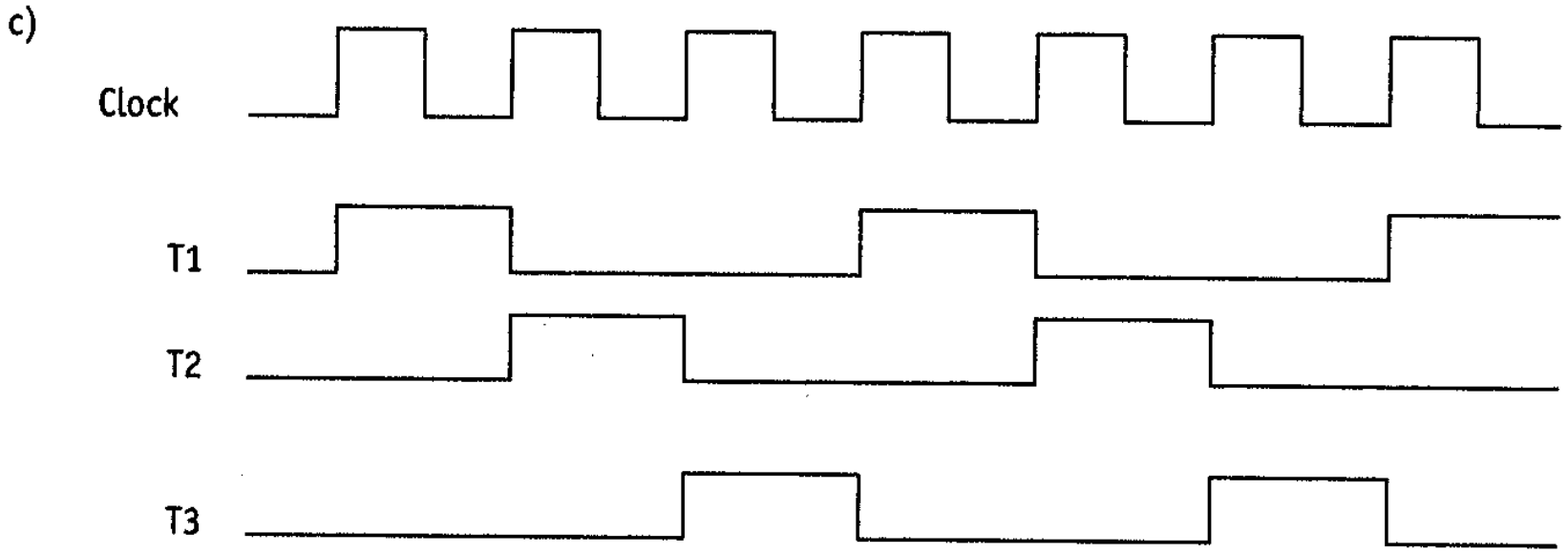
Data In is not edge triggered by LD

Incrementation is edge triggered by INC

FIGURE 5.50


Sequencer
a)
Init
Sequencer
T1
T2
T3
CK
Circuit 3
CK
Circuit 2
CK
Circuit 1

T1, T2, T3 clock subcycles



Components of H1

- Central Processing Unit (CPU)
- Clock/sequencer
- Main memory
- Keyboard
- Monitor

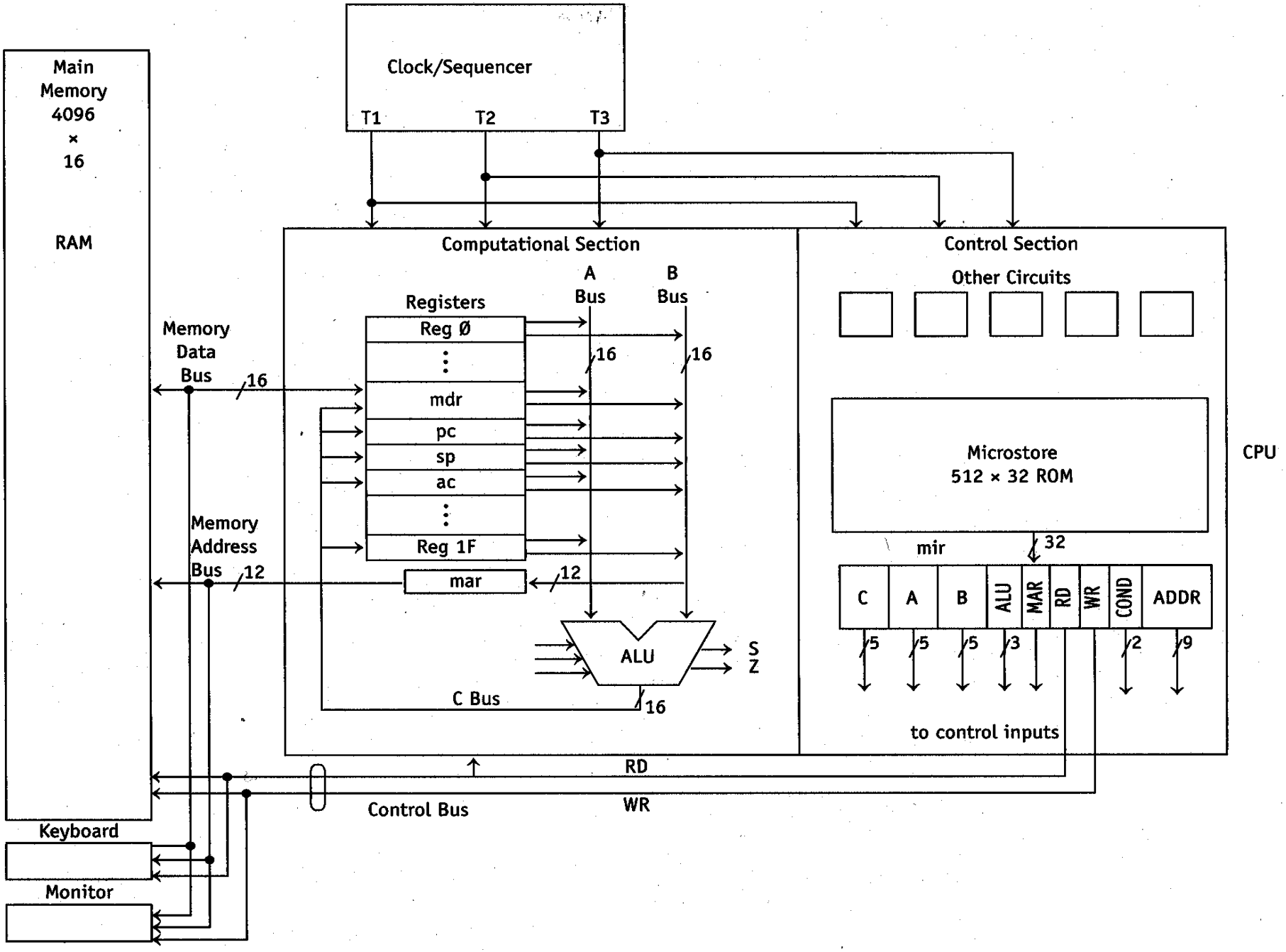
Two sections in the CPU

- *Computational section*: adds, subtracts, etc.
- *Control section*: generates signals that control the various circuits in the computer.

Computational section

- Consists of register bank, ALU, and interconnecting buses.
- Forms a circle: registers drive A and B buses; A and B buses drive ALU; ALU output drives registers.
- This circle is called the *data path*.

FIGURE 6.1


Main Memory 4096 × 16
RAM
Clock/Sequencer
T1
T2
T3
Computational Section
Control Section
Other Circuits
A Bus
B Bus
Registers
Reg Ø
mdr
pc
sp
ac
Reg 1F
mar
Memory Data Bus
16
Memory Address Bus
12
ALU
S
Z
C Bus
Microstore 512 × 32 ROM
mir
32
C
A
B
ALU
MAR
RD
WR
COND
ADDR
5
3
2
9
to control inputs
CPU
RD
WR
Control Bus
Keyboard
Monitor

To increment the pc register

- Route the number 1 and the contents of the pc register to the ALU.
- Instruct the ALU to add.
- Route the output of the ALU back into the pc register.

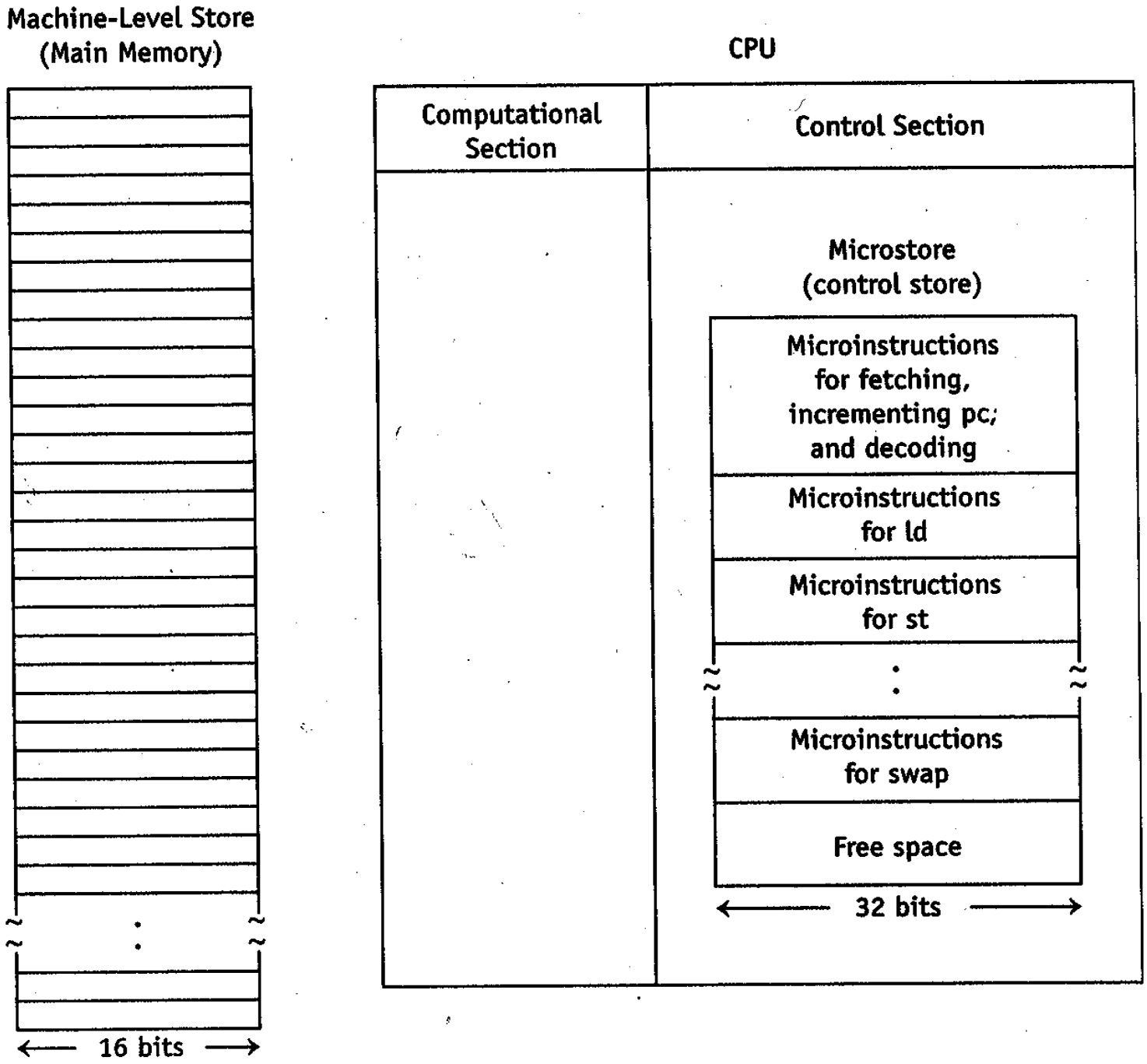
The computer has many *control inputs*. For example, the ALU has three control inputs that determine its operation.

Executing a microinstruction

- Load the mir with a microinstruction from microstore.
- The outputs of the mir then drive the various control inputs in the computer.
- One or more micro-operations occur.

Main memory is external to the CPU
Microstore is internal to the CPU

FIGURE 6.2

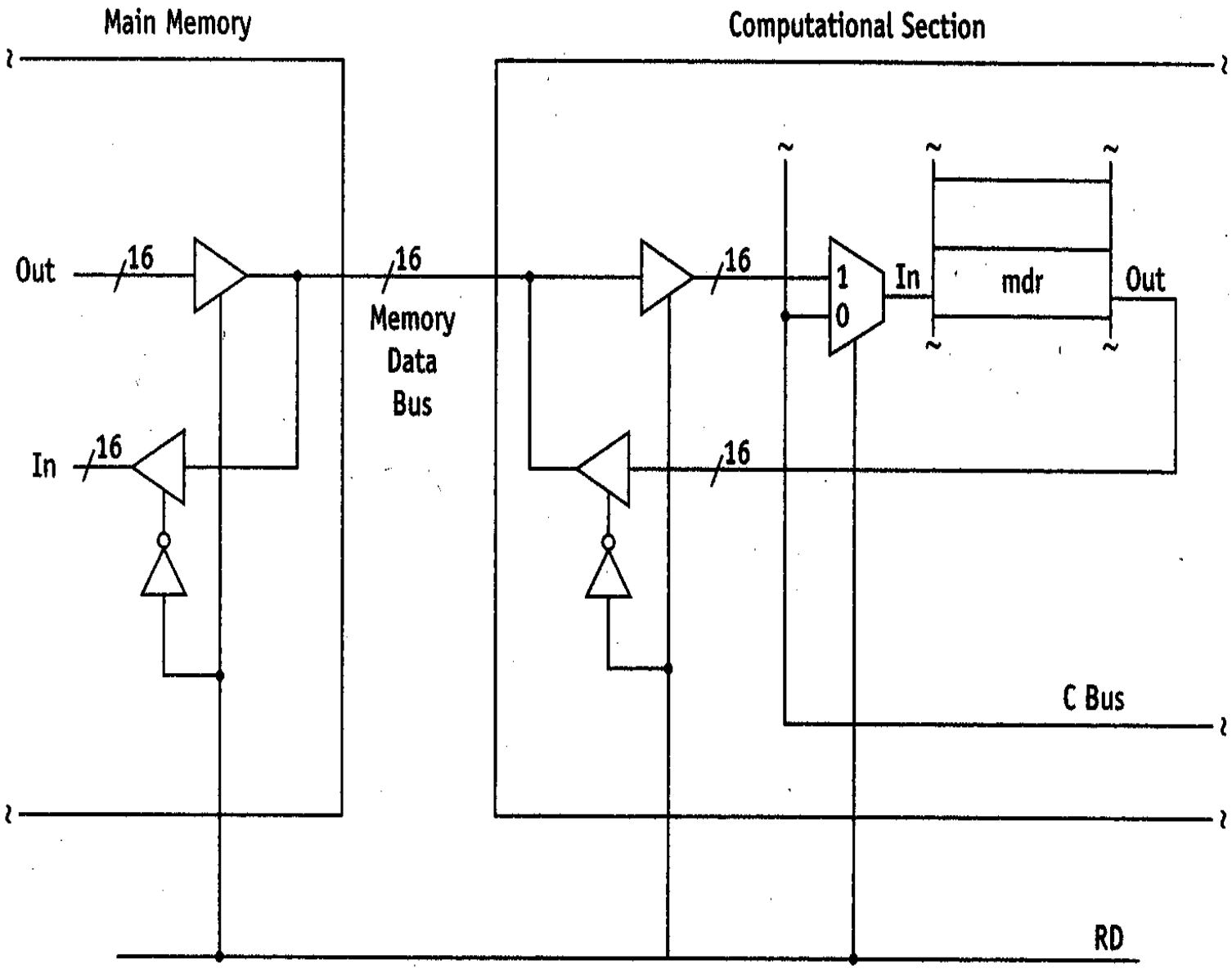

Machine-Level Store
(Main Memory)
16 bits
CPU
Computational Section
Control Section
Microstore
(control store)
Microinstructions for fetching, incrementing pc; and decoding
Microinstructions for ld
Microinstructions for st
Microinstructions for swap
Free space
32 bits

Buses

- **Memory data bus**: bidirectional bus between CPU and main memory. RD determines its direction.
- **Memory address bus**: unidirectional bus from CPU to main memory.

FIGURE 6.3

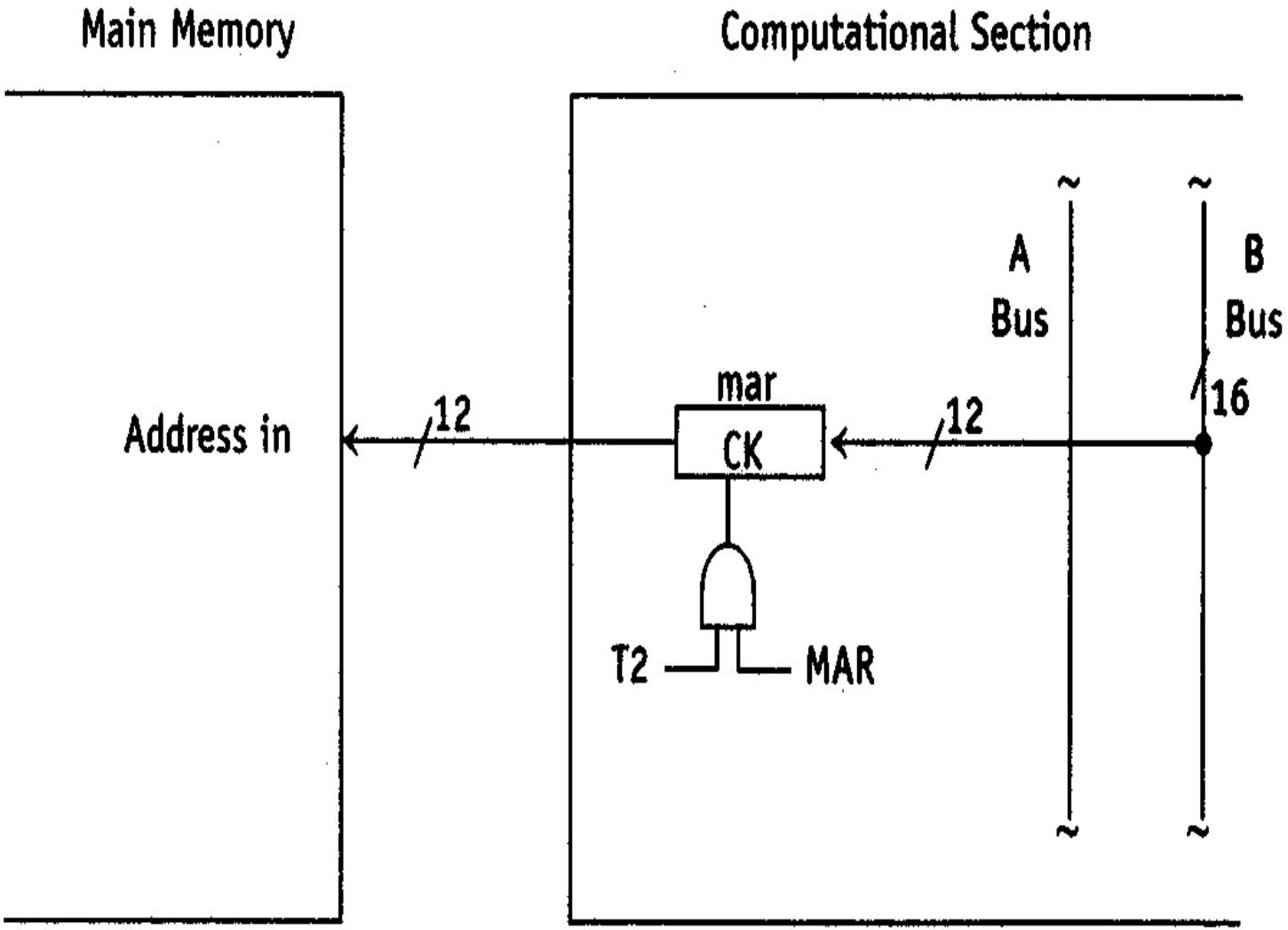
Memory Data Bus



The mar is loaded from the B bus.

The mar holds an address so it is only 12 bits wide.

FIGURE 6.4 Memory Address Bus



Control inputs on registers

- EA: enables register output to A bus
- EB: enables register output to B bus
- EC: enables loading of the register from the C bus. EC drives CK input. No EC on read-only registers.
- EA, EB, EC may be 1 simultaneously.

Naming control inputs on registers

EA, EB, EC for register 6 are called EA6, EB6, and EC6

6.2.5 Register Bank

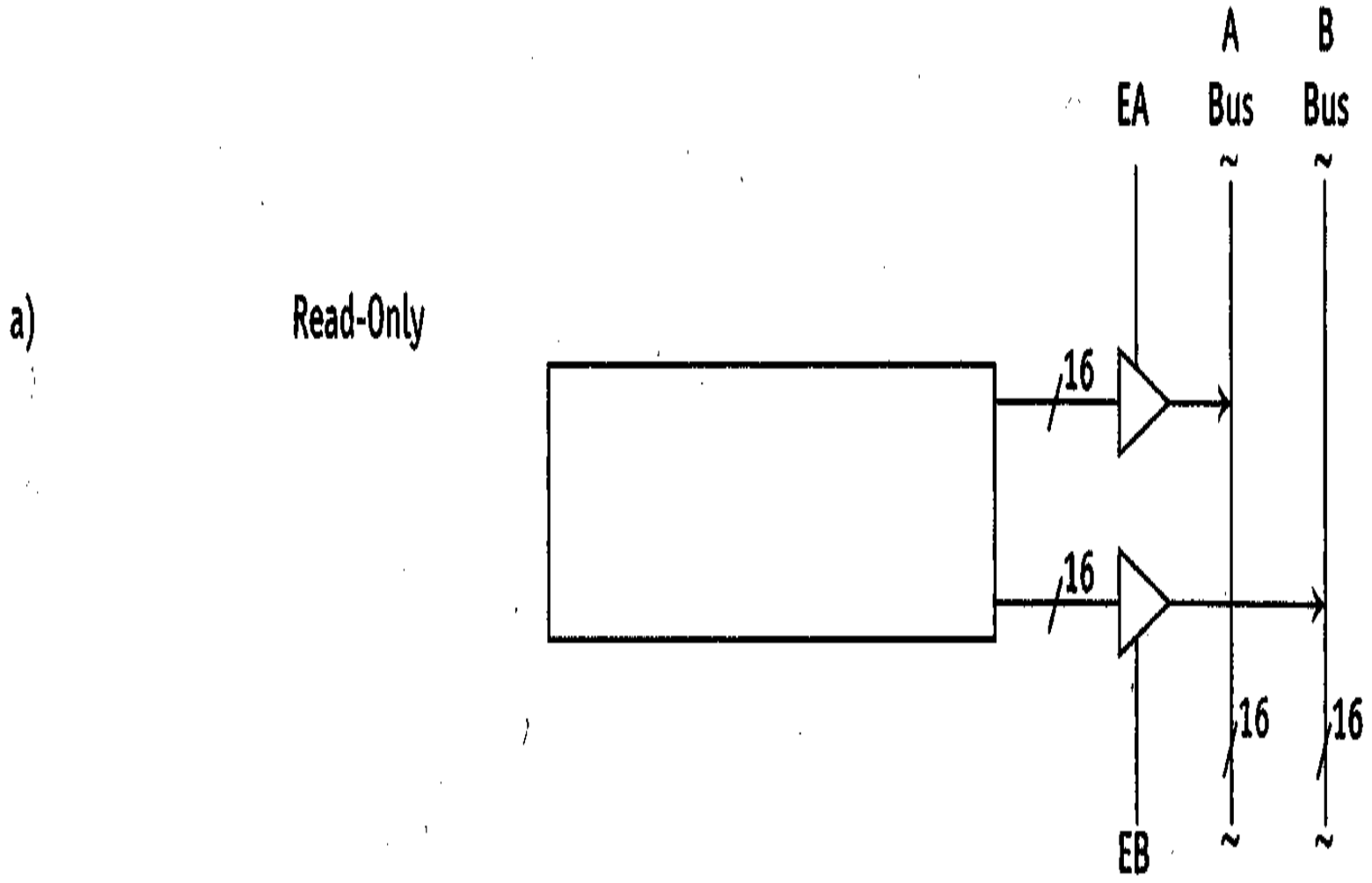
Read-only registers

The register bank in H1 consists of 32 16-bit registers numbered in hex from 0 to 1F. The first five registers are read-only registers that hold the following constants:

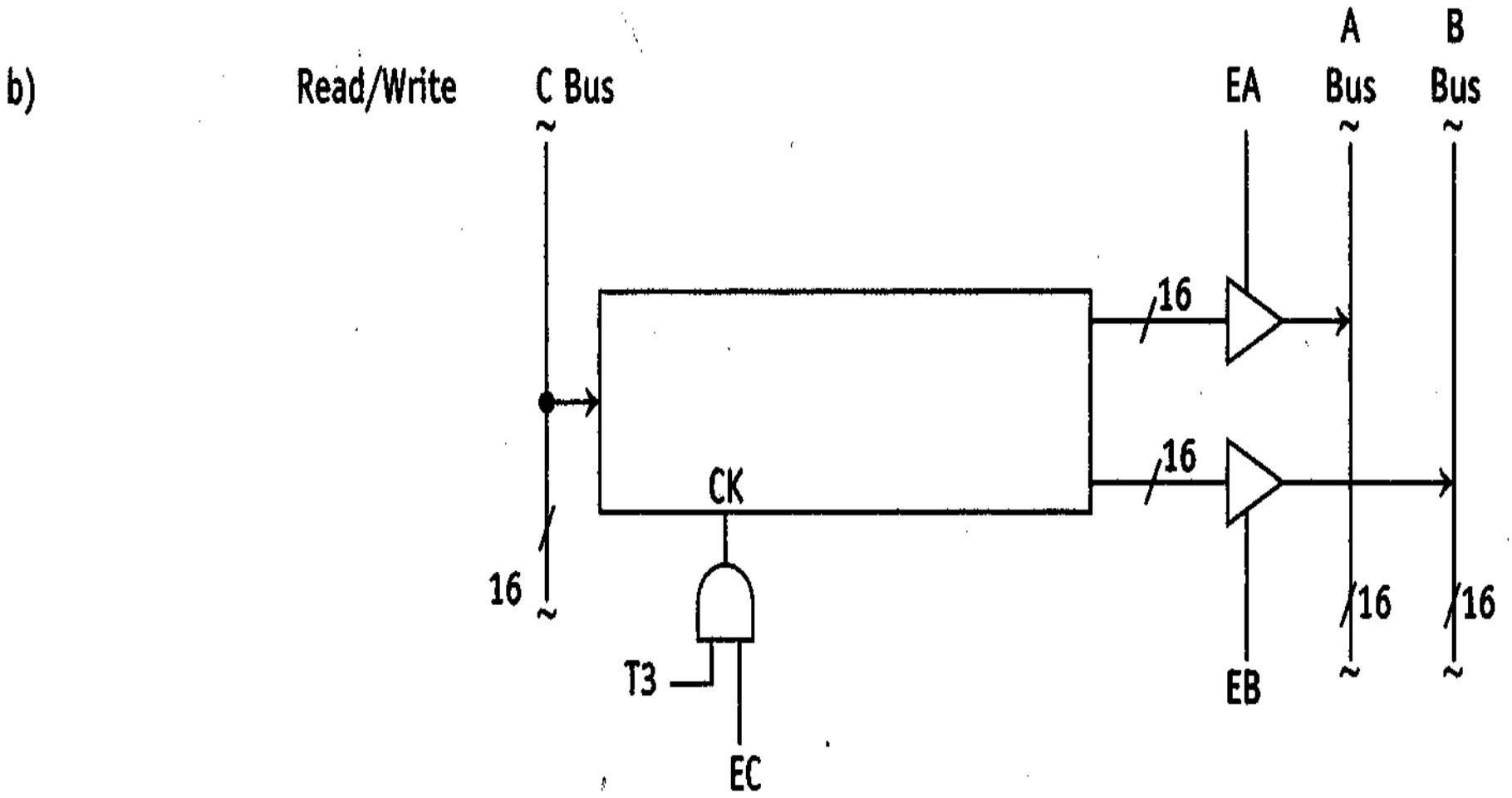
Number	Contents	Name
0	0000	
1	0001	
2	0FFF	xmask
3	00FF	ymask
4	000F	zmask

Read-only register

FIGURE 6.5



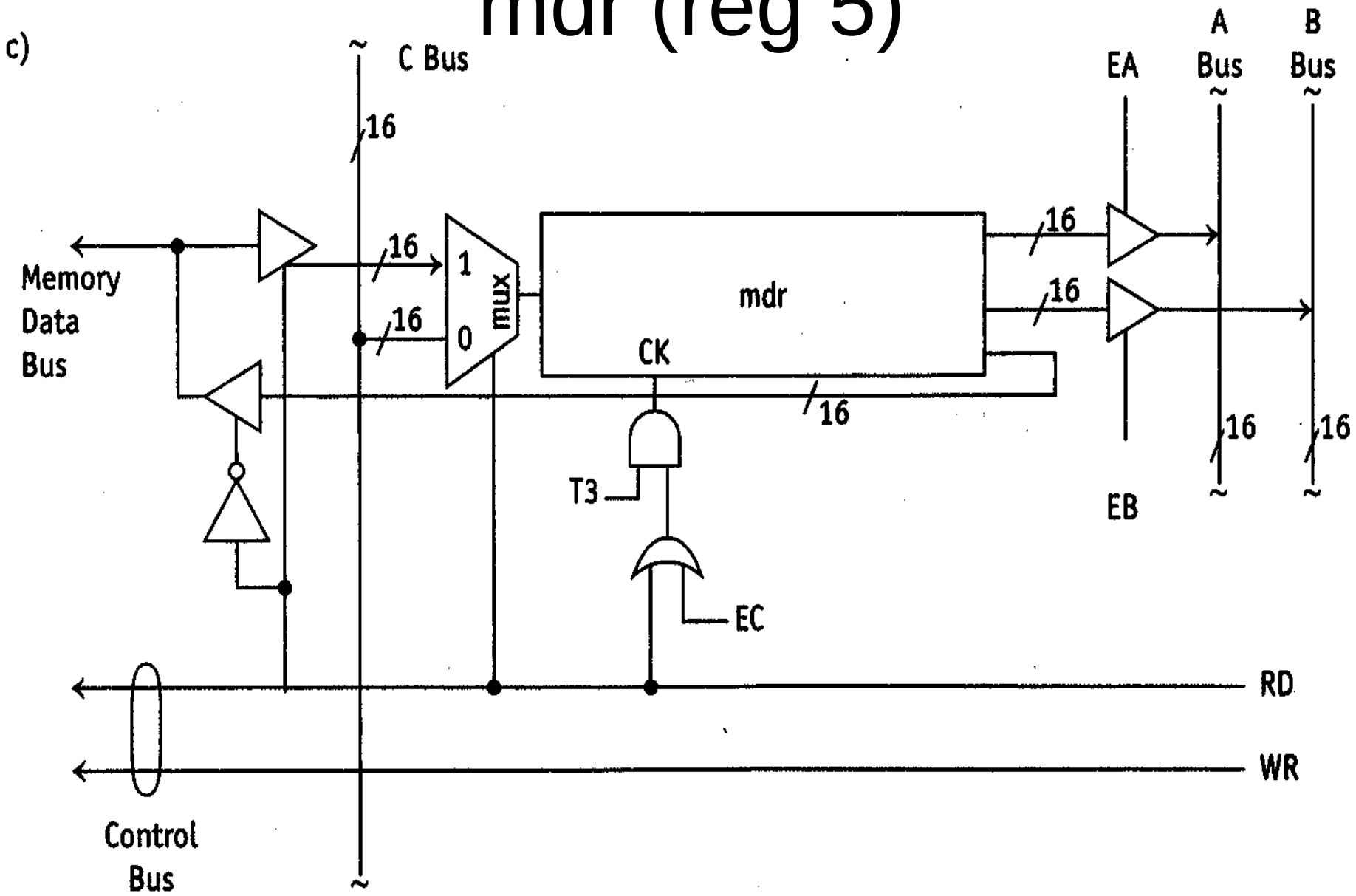
Read/write registers



memory data register (mdr) (register 5)

- More complicated than other read/write registers.
- Can be loaded from either the C bus (if RD = 0 and EC = 1) or the memory data bus (if RD = 1).
- the mdr drives the memory data bus on a memory write operation.

mdr (reg 5)


c)
C Bus
16
A
Bus
B
Bus
EA
Memory
Data
Bus
16
16
1
0
mux
mdr
16
16
CK
16
16
16
T3
EB
EC
RD
WR
Control
Bus

Control inputs needed to add 1 to the pc register (reg 6)

Control Input	Effect
EA6 = 1	Places the contents of register 6 on the A bus.
EB1 = 1	Places 1 in register 1 on the B bus.
F2 = 1	Causes the ALU to perform function 4 (add).
F1 = 0	
F0 = 0	
EC6 = 1	Enables the loading of register 6 from the C bus.

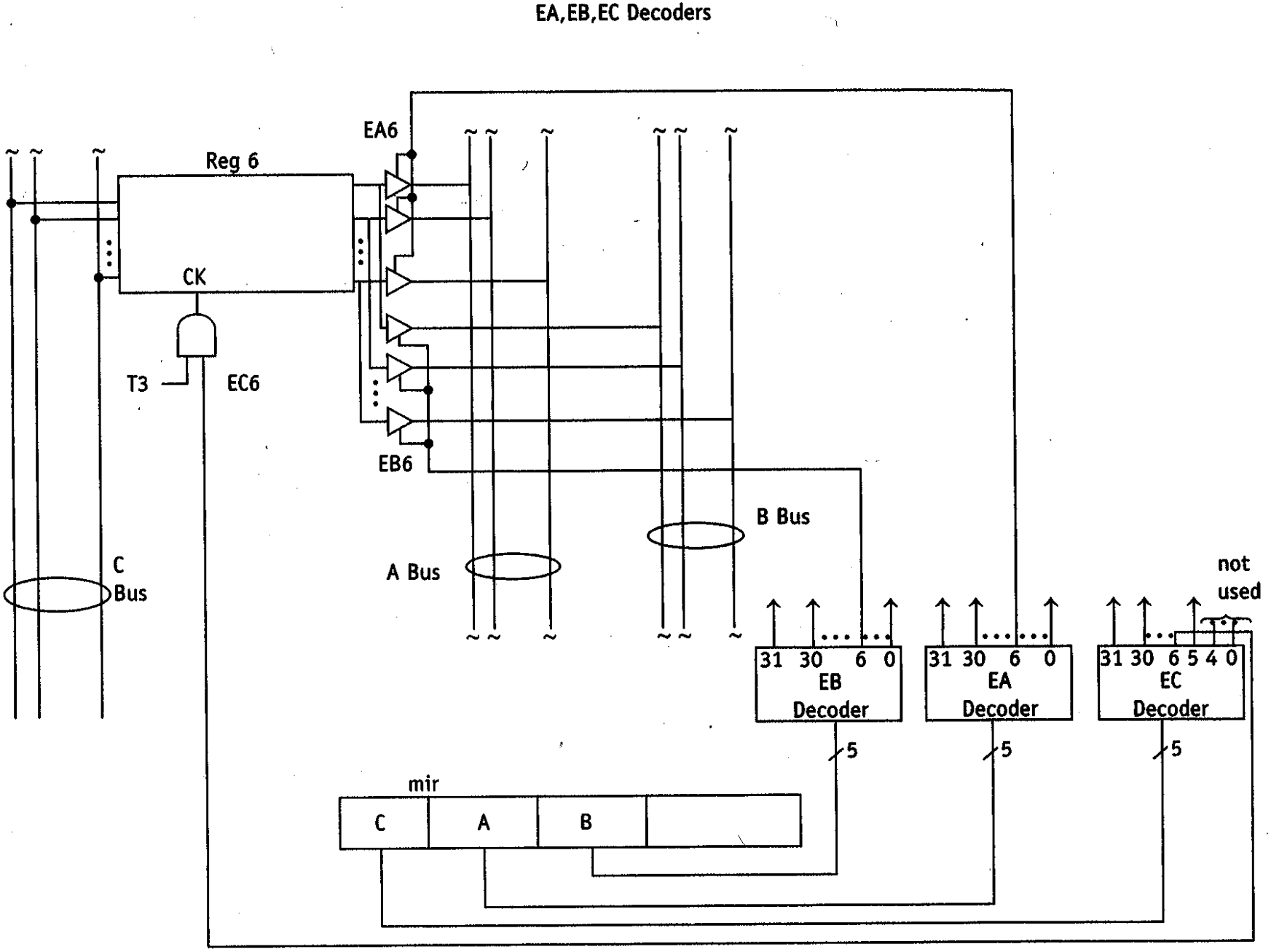
Microinstruction register (mir)

- 32 bit register that is repeatedly loaded with a microinstruction from microstore.
- Its outputs drive the various control inputs.
- A microinstruction is executed when it is loaded into the mir.

Reducing the size of a microinstruction and the mir

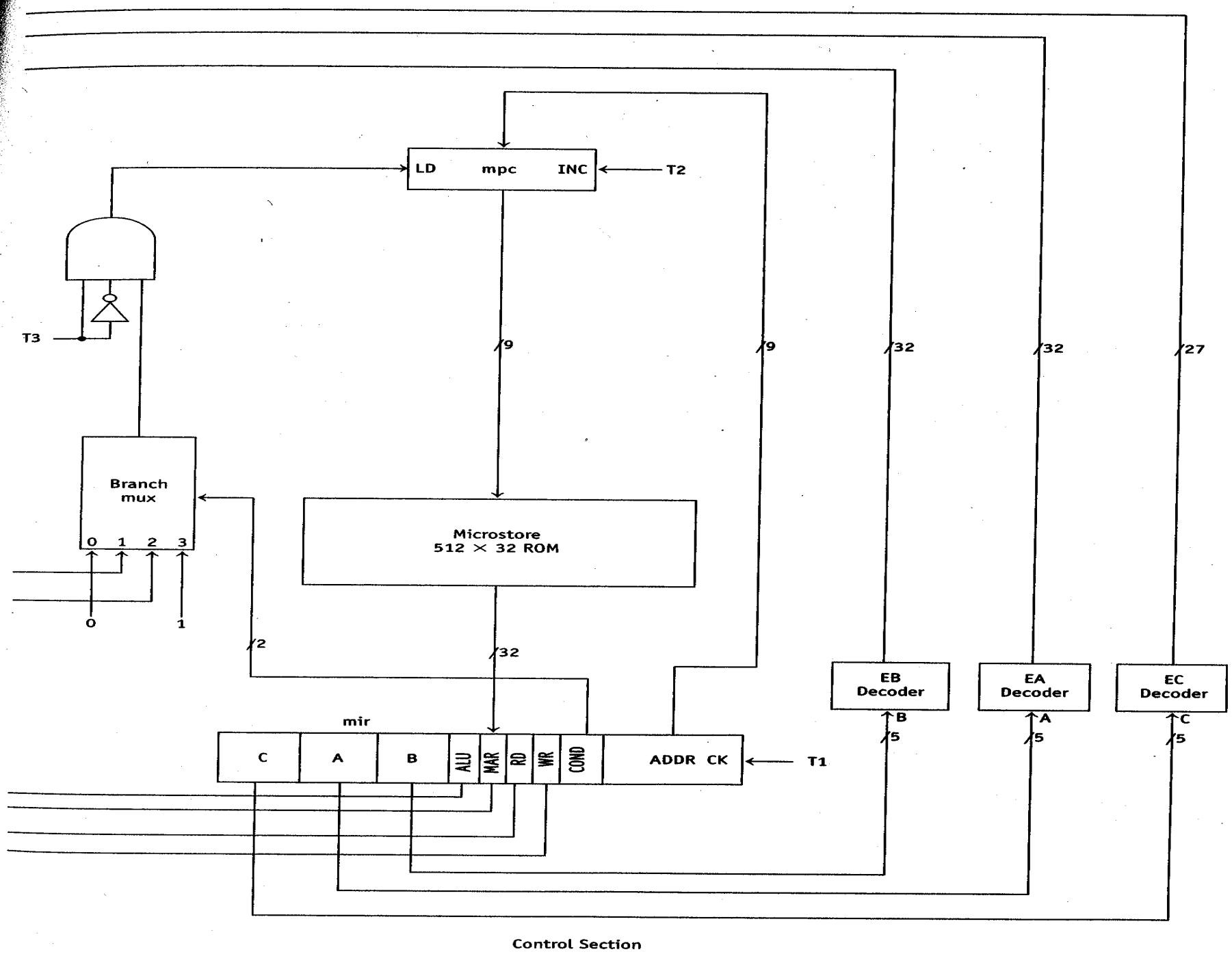
- Use decoders.
- A five-input decoder has 32 output lines– saves 32-5 = 27 bits in the microinstruction. See next slide.

FIGURE 6.6

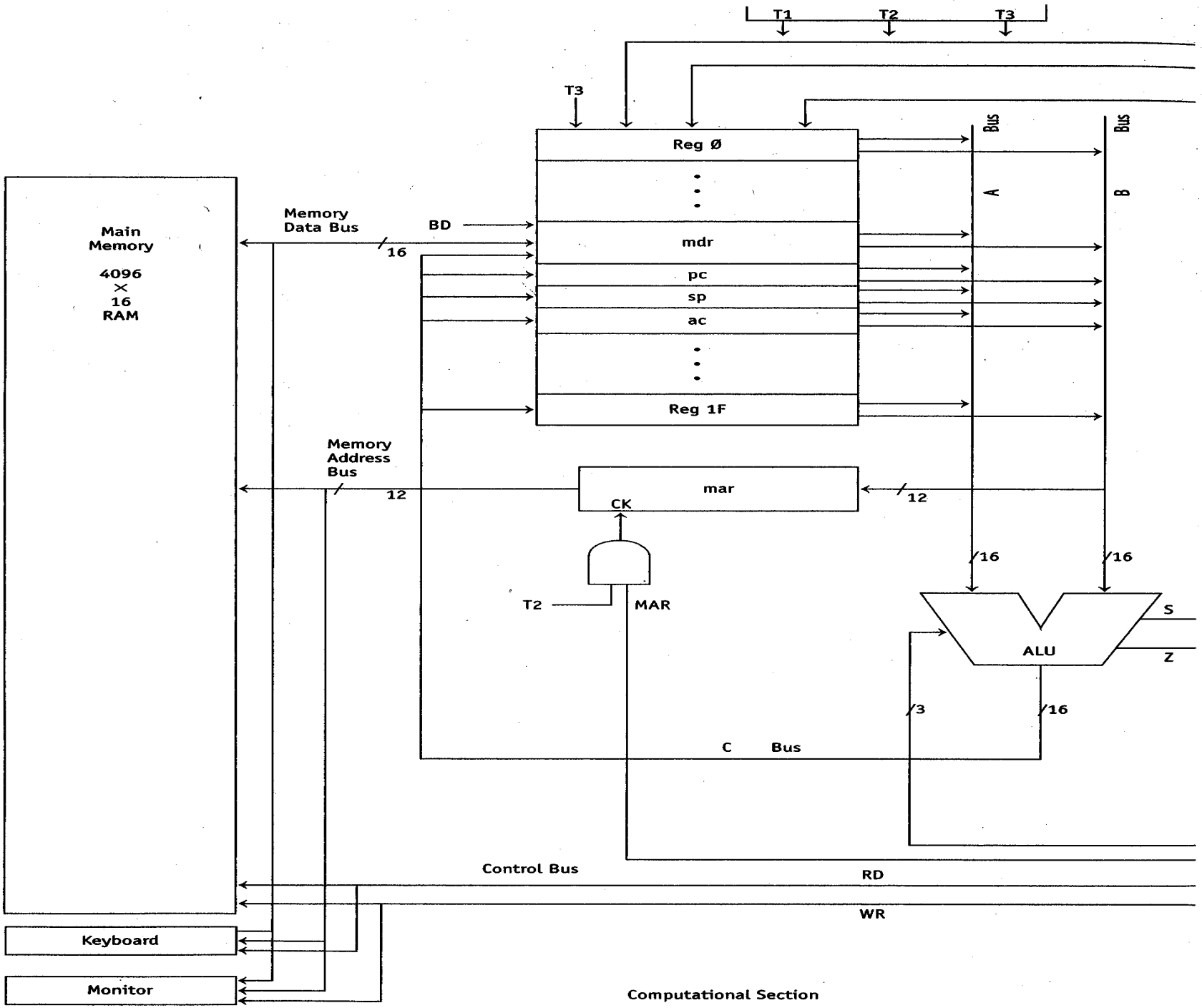

EA,EB,EC Decoders
EA6
Reg 6
CK
T3
EC6
EB6
C
Bus
A Bus
B Bus
not
used
31 30 6 0
EB
Decoder
31 30 6 0
EA
Decoder
31 30 6 5 4 0
EC
Decoder
5
5
5
mir
C
A
B

Microinstruction program counter (mpc)

- At T1, the microinstruction whose address is in the mpc is loaded into the mir. This microinstruction is then “executed” at T2 and T3.
- At T2, the mpc is incremented. Thus, on the next T1, the next microinstruction is loaded into the mir and executed.



Control Section


T1
T2
T3
T3
Reg Ø
mdr
pc
sp
ac
Reg 1F
Bus
A
Bus
B
Main Memory
4096 × 16 RAM
Memory Data Bus
BD
16
Memory Address Bus
12
mar
12
CK
T2
MAR
16
16
ALU
S
Z
3
16
C Bus
Control Bus
RD
WR
Keyboard
Monitor
Computational Section

Transferring control

- We call a transfer of control at the machine level a ***jump***.
- We call a transfer of control at the microlevel a ***branch***.
- A jump is caused by loading a new address into the pc register.
- A branch is caused by loading a new address into the mpc register.

Branching

- Every microinstruction has an ADDR field—a field that can contain a branch-to address.
- To branch, the address in ADDR of the microinstruction in the mir is loaded into the mpc at T3.
- The mpc is loaded only if the branch multiplexer outputs a 1.

FIGURE 6.8

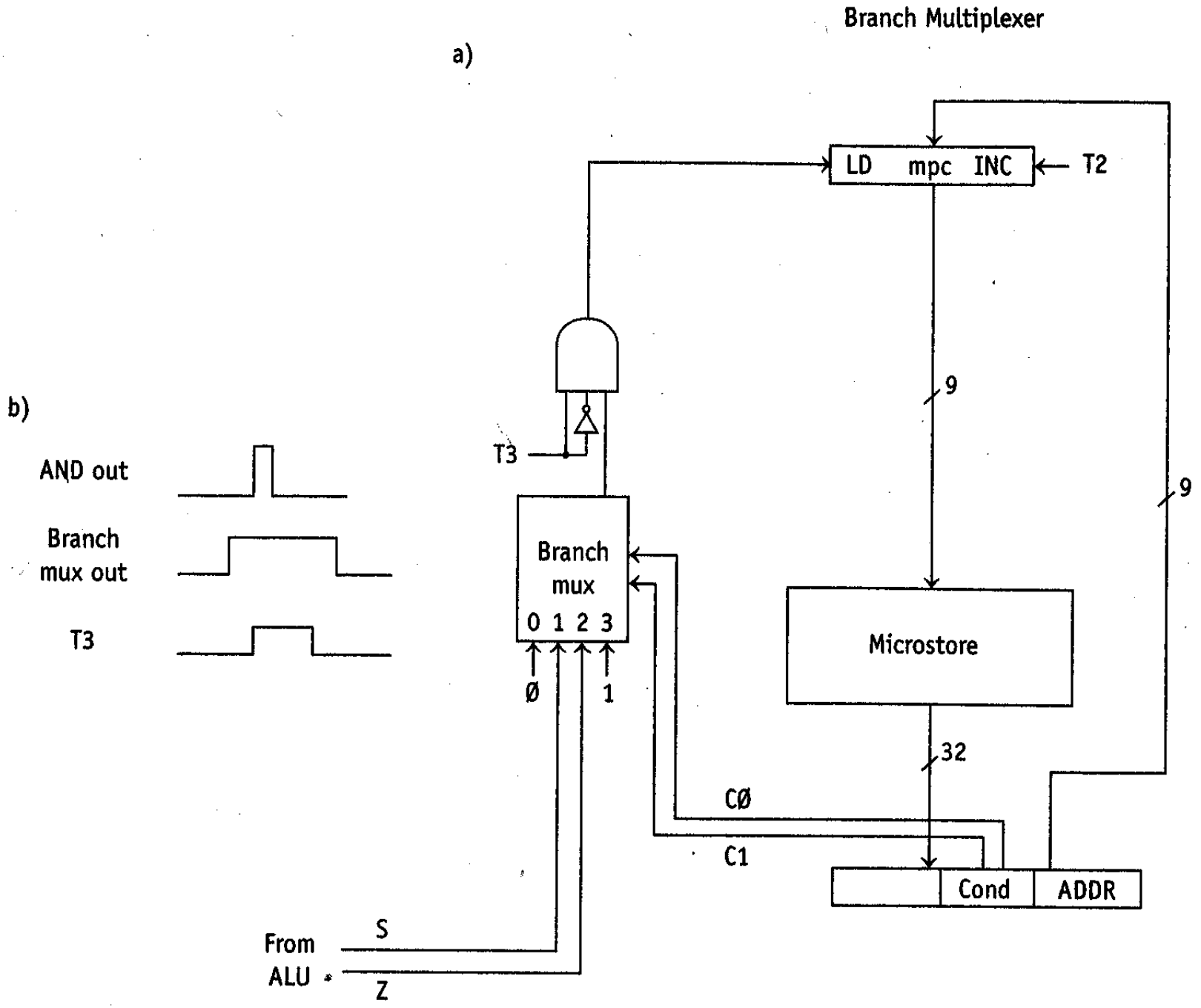

Branch Multiplexer
a)
b)
LD mpc INC
T2
9
9
T3
AND out
Branch mux out
T3
Branch mux
0 1 2 3
Ø
1
Microstore
32
CØ
C1
Cond
ADDR
From ALU
S
Z

FIGURE 6.9

Branch Mux

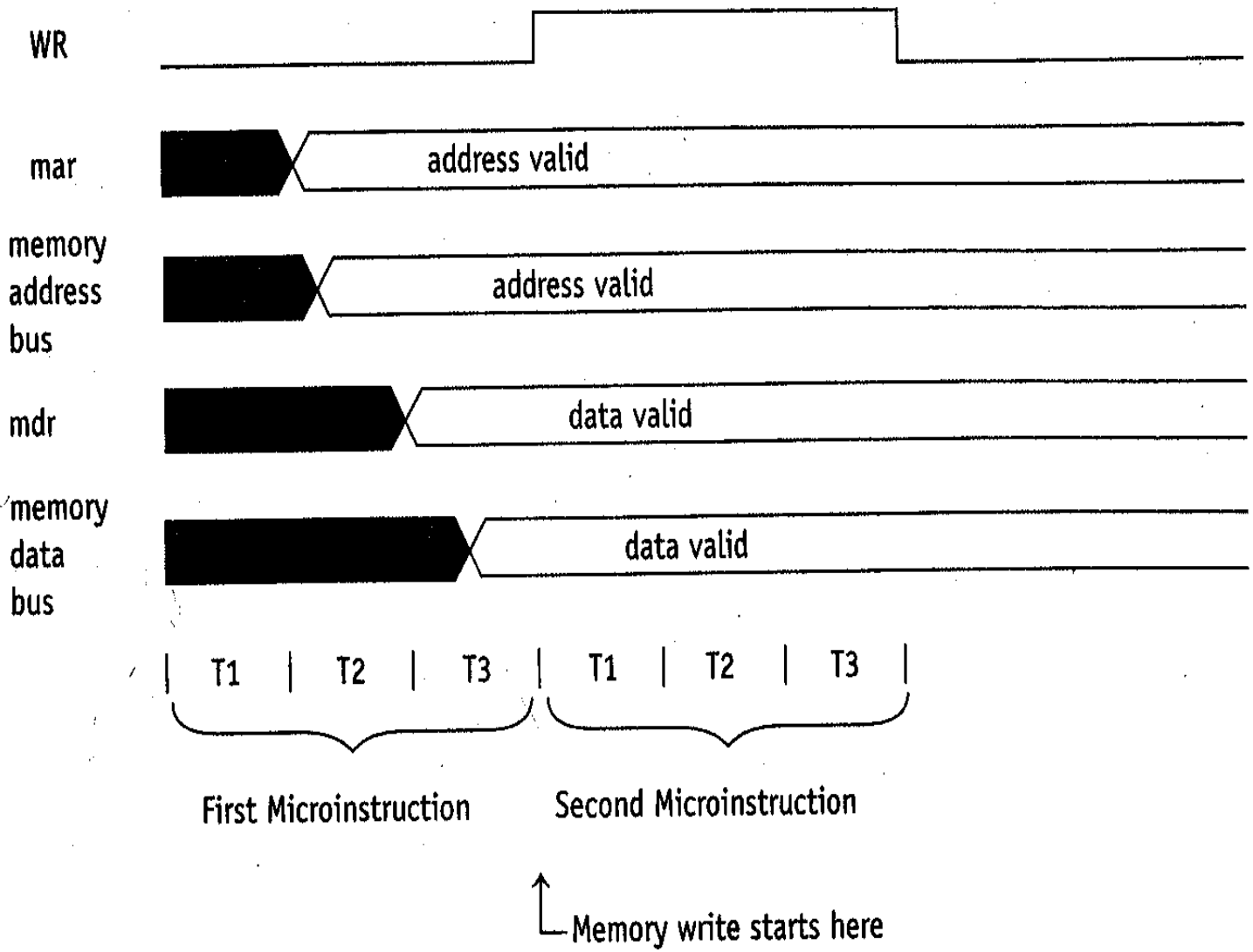
C1	C0	Output	Effect
0	0	0	No branch
0	1	S	Branch if S = 1
1	0	Z	Branch if Z = 1
1	1	1	Branch always

Reading/writing to main memory

- Reading and writing both require a two-microinstruction sequence.
- The first microinstruction loads the mar, and in the case of a write, the mdr.
- The second microinstruction triggers the read or write main memory operation by asserting RD or WR.
- The address on the memory address bus must be stable before RD or WR is asserted (which is why a two-microinstruction sequence is needed).
- The data on the memory data bus must be valid before WR is asserted.

Timing—main memory write

FIGURE 6.10 a)



Timing—main memory read

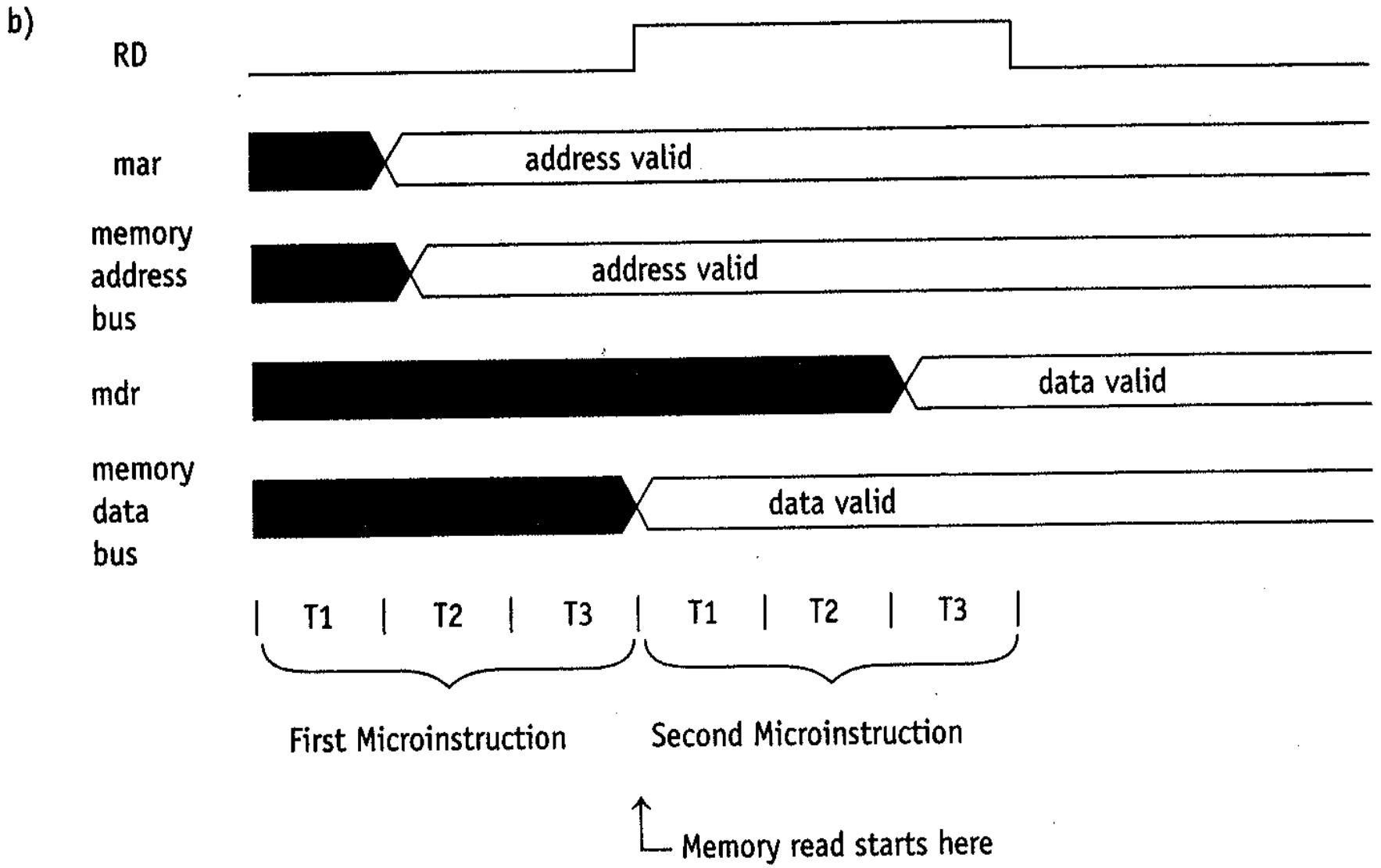


FIGURE 6.11 Microinstruction format:

C	A	B	ALU	MAR	RD	WR	COND	ADDR	
5	5	5	3	1	1	1	2	9	width (bits)

A: Register number of register outputting to A bus
B: Register number of register outputting to B bus
C: Register number of register to be loaded from C bus

ALU
0: A (left input straight through)
1: ~A (bitwise complement)
2: A & B (bitwise AND)
3: A * B
4: A + B
5: A - B
6: Shift left one position
7: Shift right one position

MAR
0: Do not load mar
1: Load mar from B bus

RD
0: Do not read
1: Read from mem[mar]

WR
0: Do not write
1: Write mdr to mem[mar]

COND
0: No branch
1: Branch if S = 1
2: Branch if Z = 1
3: Always branch
ADDR: Branch-to address

Basic instruction set

- A very simple instruction set (only 8 instructions)
- Its implementation in microcode is easy to understand.
- You will learn how to write microcode by studying the microcode for the basic instruction set. You will then be able to create new instruction sets by writing the microcode that implements them.

FIGURE 6.12 Basic Instruction Set

Opcode (hex)	Assembly Form	Name	Description
0	ld x	Load	ac = mem[x];
2	st x	Store	mem[x] = ac;
4	shll z	Shift left logical	ac << z;
6	shrl z	Shift right logical	ac >> z;
8	addc x	Add constant	ac = ac + x;
A	flip	Flip ac	ac = ~ac;
C	mult x	Multiply	ac = ac * mem[x];
E	jn x	Jump on negative	if (ac < 0) pc = x;

A microinstruction has three forms: binary, hex, and symbolic.

Binary:
00110000010011010010000000000000

Hex:
304D2000

Symbolic:
pc = 1 + pc; mar = pc;

What microcode that implements an instruction set must do

1. Fetch the instruction whose address is in the `pc` register.
2. Increment the `pc` register.
3. Decode the opcode (i.e., determine the opcode).
4. Execute the instruction.

Instruction register (ir)

When the CPU fetches a machine instruction it places it in the ir.

Decoding register (dc)

The CPU also places a copy of the machine instruction in the dc register. The dc register is used during instruction decoding.

The dc register holds the machine instruction as the instruction is shifted left in the decoding process. This shifting process corrupts the machine instruction in the dc register. However, the uncorrupted original is available in the ir.

FIGURE 6.13

```
pc = 1 + pc; mar = pc;
  C       A       B     ALU   MAR   RD   WR    COND     ADDR
00110   00001   00110   100    1     0    0     00     xxxxxxxxx

  rd;
  C       A       B     ALU   MAR   RD   WR    COND     ADDR
00000   xxxxx   xxxxx   xxx    0     1    0     00     xxxxxxxxx

  ir = mdr;
  C       A       B     ALU   MAR   RD   WR    COND     ADDR
01001   00101   xxxxx   0      0     0    0     00     xxxxxxxxx

  dc = ir; if (s) goto L1;
01010   01001   xxxxx   0      0     0    0     01     000000111
```

These two symbolic microinstructions yield the same binary microinstruction (order within one microinstruction does not matter).

pc = 1 + pc; mar = pc;

mar = pc; pc = 1 + pc;

pc = 1 + pc; mar = pc;

When is the pc register incremented?

The mar is loaded at T2. The pc is loaded at T3. Thus, the mar is loaded from the pc before the pc is incremented.

Writing microcode

- We write microcode in symbolic form.
- But H1 needs the binary form.
- So we translate the symbolic form to binary using the *microassembler* **has**.
- ".has" extension is for symbolic microcode
- ".hor" extension is for binary microcode
- Hex form is a shorthand representation of the binary form.

FIGURE 6.14

```
 1 /    Basic Instruction Set Horizontal Microcode    b.has
 2
 3 /    The label indicates the opcode decoded up to that point.
 4 /    For example, control reaches the label L011 when opcode
 5 /    bits 011 have been decoded.
 6
 7 /*************************************************************/
 8 /       Fetch instruction and increment pc register          /
 9 /*************************************************************/
10 fetch: mar = pc; pc = pc + 1;                  / increment pc
11        rd;                                     / fetch mach inst
12        ir = mdr;                               / put mach inst in ir
13 /*************************************************************/
14 /       Decode opcode                                        /
15 /*************************************************************/
16        dc = ir; if(s) goto L1                  / test 1st bit
17 L0:    dc = left(dc); if (s) goto L01;         / test 2nd bit
18 L00:   dc = left(dc); if (s) goto L001;        / test 3rd bit
19        goto L000;
20 L1:    dc = left(dc); if (s) goto L11;         / test 2nd bit
21 L10:   dc = left(dc); if (s) goto L101;        / test 3rd bit
22        goto L100;
23 L01:   dc = left(dc); if (s) goto L011;        / test 3rd bit
24        goto L010;
25 L11:   dc = left(dc); if (s) goto L111;        / test 3rd bit
26             goto L110:
27 /*************************************************************/
28 /       Microcode for each instruction                       /
29 /*************************************************************/
30 L000:  /--------------------------  LD  ---------------------------
```

(continued)

FIGURE 6.14 (continued)

```
31          mar = ir;                                   / get add from inst
32          rd;                                         / read operand
33          ac = mdr; goto fetch;                       / load ac with operand
34  L001:   /-------------------------- ST ---------------------------
35          mar = ir; mdr = ac;                         / prepare for write
36          wr; goto fetch;                             / write
37  L010:   /-------------------------- SHLL ---------------------------
38          f = ir & zmask; if (z) goto fetch;          / get z-field
39  lloop:  ac = left(ac);                              / shift ac left
40          f = f - 1; if (z) goto fetch;               / dec ac and test
41          goto lloop;                                 / branch to lloop
42  L011:   /-------------------------- SHRL ---------------------------
43          f = ir & zmask; if (z) goto fetch;          / get z-field
44  rloop:  ac = right(ac);                             / shift ac right
45          f = f - 1; if (z) goto fetch;               / dec ac and test
46          goto rloop;                                 / branch to rloop
47  L100:   /-------------------------- ADDC ---------------------------
48          f = ir & xmask;                             / get const from x-field
49          ac = ac + f; goto fetch;                    / add const to ac
50  L101:   /-------------------------- FLIP ---------------------------
51          ac = ~ac; goto fetch                        / bitwise compl ac
52  L110:   /-------------------------- MULT ---------------------------
53          mar = ir;                                   / get add from inst
54          rd;                                         / read operand
55          ac = ac * mdr; goto fetch;                  / mult ac by operand
56  L111:   /-------------------------- JN ---------------------------
57          0 = ac; if (s) goto dojn;                   / branch if ac neg
58          goto fetch;                                 / branch
59  dojn:   pc = ir & xmask; goto fetch;                / put new add in pc
```

Multiple shifts are inefficient with the simple one-position shifter.
Better: use a barrel shifter.

H1 has a barrel shifter but it is not the default shifter.

Configuration file

To use new microcode, you must provide a configuration file that describes the microcode. The configuration file is used by **mas**, **has** (the microassembler), and **sim**.

FIGURE 6.16

```
;                                                          b.cfg
;       Configuration file for the basic instruction set

          ; Part 1 (mnemonics in opcode order)
ld      1       ; mnemonic and number of operands for opcode 0
?
st      1       ; mnemonic and number of operands for opcode 2
?
shll    1       ; mnemonic and number of operands for opcode 4
?
shrl    1       ; mnemonic and number of operands for opcode 6
?
addc    1       ; mnemonic and number of operands for opcode 8
?
flip    0       ; mnemonic and number of operands for opcode A
?
mult    1       ; mnemonic and number of operands for opcode C
?
jn              ; mnemonic and number of operands for opcode E

%%          ; Part 2 (reg names in number order starting with reg 6)
pc              ; register 6
?               ; not using register 7
ac              ; register 8
ir              ; register 9
dc              ; register A
                ; remaining regs can be referenced by their hex numbers

%%          ; Part 3 (special addresses and numbers)
0               ; horizontal start fetch address (hex)
0               ; vertical start fetch address (hex)

0               ; horizontal increment pc address (hex)
3               ; vertical increment pc address (hex)

1               ; horizontal read machine instruction address (hex)
1               ; vertical read machine instruction address (hex)

6               ; pc register number (hex)
0               ; sp register number (hex)
8               ; I/O register number (hex)

6               ; machine-level register numbers (hex)
8
```

!-directive

- Placed at the beginning of a machine-level assembly language program.
- Indicates the microcode/configuration file to use.
- For example, !b indicates that b.hor (microcode file) and b.cfg (configuration file) should be used.

bprog.mas—a test program for the basic instruction set microcode.

FIGURE 6.17

```
 1            !b                    bprog.mas
 2 start:     ld       x          ; load from x
 3            flip                ; flip bits in ac
 4            addc     3          ; add the constant 3
 5            mult     y          ; multiply ac by y
 6            st       z          ; store product in z
 7            shll     2          ; shift left logical 2
 8            shrl     2          ; shift right logical 2
 9            jn     start        ; jump if ac is negative
10            dout                ; output ac in decimal
11            halt
12 x:         dw       -1
13 y:         dw       2
14 z:         dw       0
```

Using new microcode b.has with configuration file b.cfg

- Assemble microcode: has b
  (translates b.has to b.hor). has responds with
  Reading configuration file b.cfg
- Assemble test program: mas bprog
  (translates bprog.mas to bprog.mac). mas responds with
  Reading configuration file b.cfg
- Run bprog on sim: sim bprog
  sim responds with
  Reading configuration file b.cfg
  Reading microcode file b.hor

Running bprog.mac on sim.

We will examine its execution at both the machine and microlevels.

```
Starting session.  Enter h or ? for help.
---- [T7] 0: ld   /0 00A/
```

Let's run the program to completion by entering **g** (to go) to see if it works correctly. **sim** displays

```
  0: ld   /0 00A/ ac=0000/FFFF
  1: flip /A 000/ ac=FFFF/0000
  2: addc /8 003/ ac=0000/0003
  3: mult /C 00B/ ac=0003/0006
  4: st   /2 00C/ m[00C]=0000/0006
  5: shll /4 002/ ac=0006/0018
  6: shrl /6 002/ ac=0018/0006
  7: jn   /E 000/
  8: dout /FFFD / 6
  9: halt /FFFF /
Machine inst count =     A (hex) =      10 (dec)
```

```
---- [T7] o
Starting session.  Enter h or ? for help.
---- [T7] 0: ld   /0 00A/ mc
Machine-level display mode + counts
---- [T7] 0: ld   /0 00A/ g
  0: ld   /0 00A/ ac=0000/FFFF   10t
  1: flip /A 000/ ac=FFFF/0000   7t
  2: addc /8 002/ ac=0000/0003   9t
  3: mult /C 00B/ ac=0003/0003   10t
  4: st   /2 00C/ m[00C]=0000/0003   8t
  5: shl1 /4 002/ ac=0004/0018   13t
  6: shr1 /6 002/ ac=0010/0006   12t
  7: jn   /E 000/  8t
  8: dout /FFFD / 6 2t
  9: halt /FFFF /   2t
Machine inst count =     A (hex) =      10 (dec)
---- [T7]
```

To trace microcode, H1 must be *enabled*. Debugger commands then become case sensitive: uppercase for the machine-level; lower case for the microlevel.

```
---- [T7] o
Starting session.  Enter h or ? for help.
---- [T7] 0: ld   /0 00A/ enable
Microlevel enabled
```

T2 executes 2 machine instructions

```
---- [T7] 0: ld   /0 00A/ T2
  0: ld   /0 00A/ ac=0000/FFFF 10t
  1: flip /A 000/ ac=FFFF/0000 7t
---- [T2] 2: addc /8 002/
```

Lower case m changes display mode to microlevel

```
---- [T2] 2: addc /8 002/ m

Microlevel display mode

---- [T1] 0: pc = 1 + pc; mar = pc; /
```

Upper case T1 executes 1 machine instruction. When in micro display mode, all the microlevel activity during the execution of this machine instruction is traced.

The default command when in micro display mode is T1.

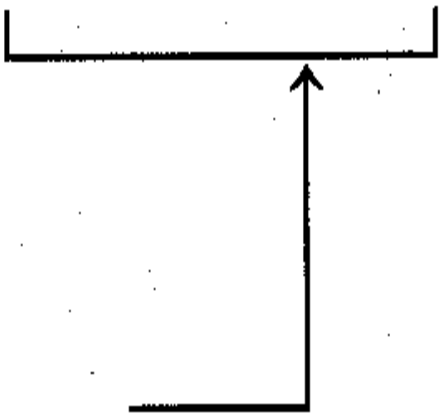
```
---- [T1] 0: pc = 1 + pc; mar = pc; / ←hit ENTER
  0: pc = 1 + pc; mar = pc;
     mar=001/002  pc=0002/0003
  1: rd;
     Rd from m[002] mdr=A000/8003  2: addc /8 003/
  2: ir = mdr;
     ir=A000/8003
  3: dc = ir; if (s) goto 7;
     dc=8000/8003
  7: dc = left(dc); if (s) goto C;
     dc=8003/0006
  8: dc = left(dc); if (s) goto 1D;
     dc=0006/000C
  9: goto 1B;
 1B: f = ir & xmask;
     f=0000/0003
 1C: ac = ac + f; goto 0;
     ac=0000/0003
---- [T1] 0: pc = 1 + pc; mar = pc; /
```

Trace shows machine instruction fetched

1: rd;

Rd from m[002] mdr=A000/8003

2: addc /8 003/



Lower case t3 executes 3 microinstructions

```
---- [T1] 0: pc = 1 + pc; mar = pc; / t3
  0: pc = 1 + pc; mar = pc;
     mar=002/003  pc=0003/0004
  1: rd;
     Rd from m[003] mdr=8003/C00B  3: mult /C 00B/
  2: ir = mdr;
     ir=8003/C00B
---- [t3] 3: dc = ir; if (s) goto 7; /
```

Lower case u0 unassembles microcode in microstore starting from location 0.

```
---- [t3] 3: dc = ir; if (s) goto 7; / u0
      C       A       B       ALU    MAR  RD   WR  COND  ADDR
/  0: pc      1       pc      +      1    0    0   0     0
   0:     pc = 1 + pc; mar = pc;

/  1: 0       0       0       0      0    1    0   0     0
   1:     rd;

/  2: ir      mdr     0       0      0    0    0   0     0
   2:     ir = mdr;

-------------------------------------------------------------

/  3: dc      ir      0       0      0    0    0   1     7
   3:     dc = ir; if (s) goto 7;

---- [t3] 3: dc = ir; if (s) goto 7; /
```

Upper case U* unassembles machine-level code in main memory

```
---- [t3] 3: dc = ir; if (s) goto 7; / U*
  0: ld   /0 00A/ flip /A 000/ addc /8 003/ mult /C 00B/
  4: st   /2 00C/ shll /4 002/ shrl /6 002/ jn   /E 000/
  8: dout /FFFD / halt /FFFF / halt /FFFF / ld   /0 002/
  C: ld   /0 000/
---- [t3] 3: dc = ir; if (s) goto 7; /
```

Lower case d* displays all of microstore in use.

Upper case D* displays all of main memory in use.

```
---- [t3] 3: dc = ir; if (s) goto 7; / d*
  0: 304D2000    00001000    49400000    52400207
  4: 5281820A    52818211    0000060E    5281820C
  8: 5281821D    0000061B    52818217    00000613
  C: 52818221    0000061E    00122000    00001000
 10: 41400600    2A122000    00000E00    7A488400
 14: 42018000    7BC34400    00000614    7A488400
 18: 4201C000    7BC34400    00000618    7A448000
 1C: 421F0600    42004600    00122000    00001000
 20: 420AC600    02000223    00000600    32448600
---- [t3] 3: dc = ir; if (s) goto 7; /
```

And to dump all machine code, enter **D***:

```
---- [t3] 3: dc = ir; if (s) goto 7; / D*
  0: 000A A000 8003 C00B 200C 4002 6002 E000    ........
  8: FFFD FFFF FFFF 0002 0000                   .....
---- [t3] 3: dc = ir; if (s) goto 7; /
```

Lower case r* displays all registers.

To show the contents of all registers, enter **r***:

```
---- [t3] 3: dc = ir; if (s) goto 7; / r*
     mpc   = 003      mar   = 003
     0     = 0000     1     = 0001     xmask = 0FFF     ymask = 00FF
     zmask = 000F     mdr   = C00B     pc    = 0004     ?     = 0000
     ac    = 0003     ir    = C00B     dc    = 000C     b     = 0000
     c     = 0000     d     = 0000     e     = 0000     f     = 0003
     10    = 0000     11    = 0000     12    = 0000     13    = 0000
     14    = 0000     15    = 0000     16    = 0000     17    = 0000
     18    = 0000     19    = 0000     1a    = 0000     1b    = 0000
     1c    = 0000     1d    = 0000     1e    = 0000     1f    = 0000
---- [t3] 3: dc = ir; if (s) goto 7; /
```

Upper case R* shows machine-level registers

```
---- [t3] 3: dc = ir; if (s) goto 7; / R*
     pc     = 0004      ac     = 0003
---- [t3] 3: dc = ir; if (s) goto 7; /
```

Upper case M returns to machine-level display

Now back to machine-level display

```
---- [t3] 3: dc = ir; if (s) goto 7; / M
Machine-level display mode + counts
---- [T2] 3: mult /C 00B/
```

N turns off trace (N is always case insensitive).

```
---- [T2¶ 3: mult /C 00B/ n
No display mode
---- [T1]
```

Finally, let's complete the execution of the program by entering **g**:

```
---- [T1] g
6
Machine inst count =      A (hex) =       10 (dec)
Micro   inst count =     51 (hex) =       81 (dec)
---- [T1]
```

To write more efficient microcode, interleave instruction execution with decoding

FIGURE 6.18 a)

Inefficient way

```
                .
                .
L000:       dc = left(dc); if (s) goto L0001;
            goto L0000;
                .
                .
                .
L0000:      /------------- LD ------------
            mar = ir;
            rd;
            ac = mdr; goto fetch;

L0001:      /------------- ST ------------
            mar = ir; mdr = ac;
            wr; goto fetch;
```

b)

More efficient way

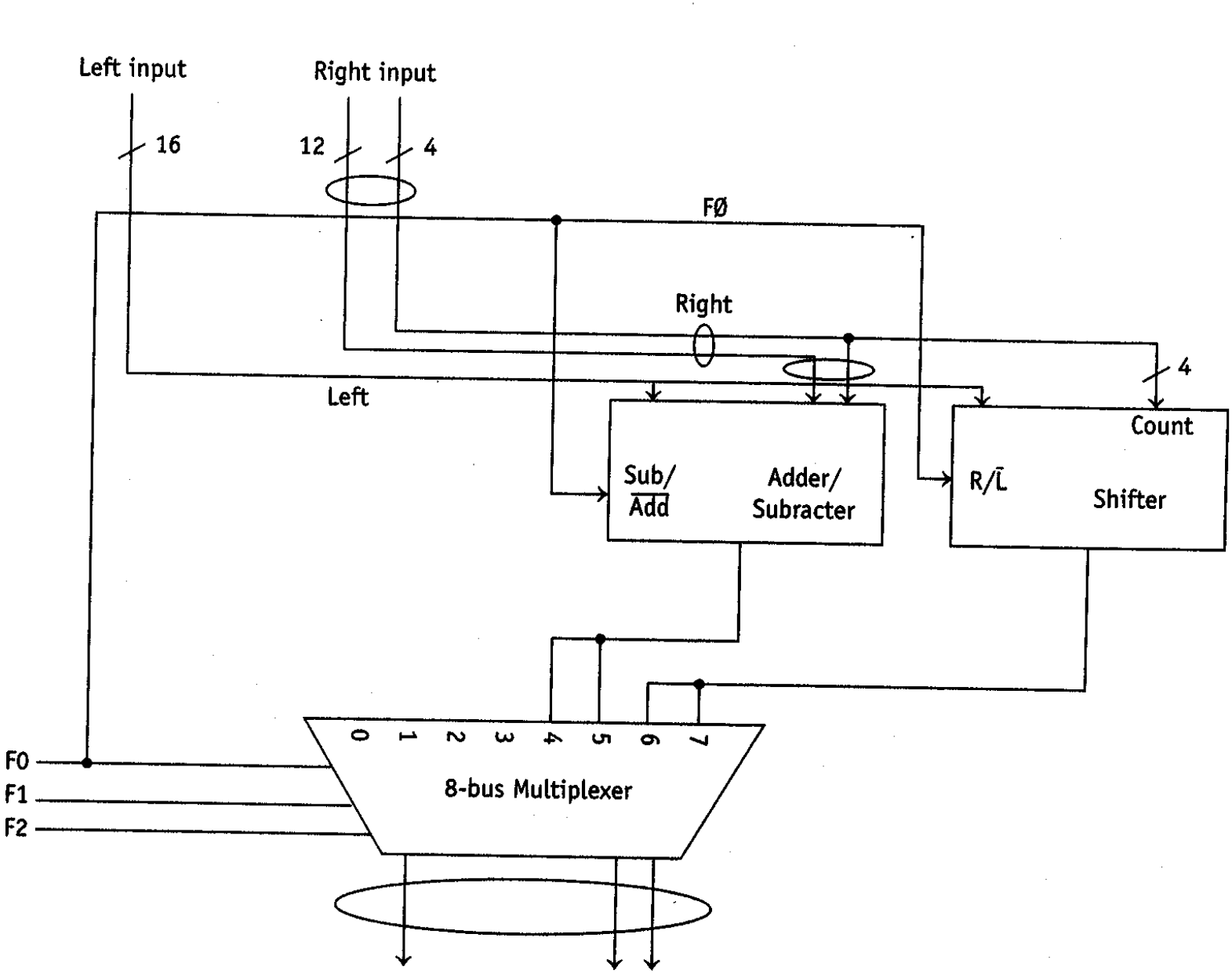
```
                .
                .
                .
L000:       dc = left(dc); if (s) goto L0001;

L0000:      /-------------- LD -------------
            mar = ir;
            rd;
            ac = mdr; goto fetch;
                .
                .
                .
L0001:      /-------------- ST -------------
            mar = ir; mdr = ac;
            wr; goto fetch;
```

Using a barrel shifter

- Specify two register operands for the left or right operations.
- Simple shifter: ac = left(ac);
- Barrel shifter: ac = left(ac, ir);
  Then the rightmost 4 bits of ir provide shift count.

FIGURE 6.15

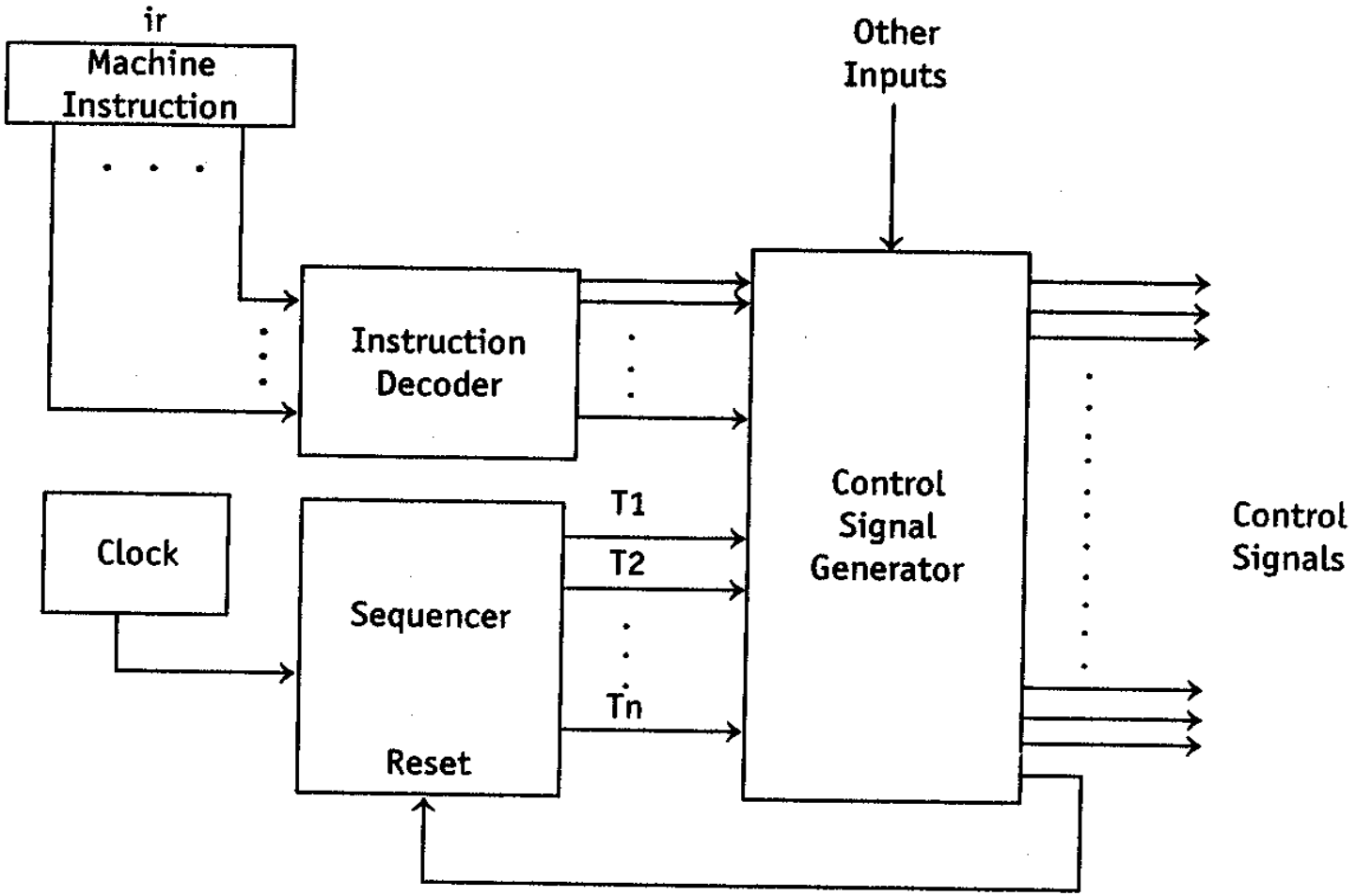

Barrel Shifter
Left input
Right input
16
12
4
FØ
Right
Left
4
Count
Sub/
Add
Adder/
Subracter
R/L
Shifter
0
1
2
3
4
5
6
7
F0
F1
F2
8-bus Multiplexer

Hardwired control

- Less flexible than microcoded control.
- Generally faster than microcoded control because does not involve the fetching of microinstructions.
- Can be very costly for a complex instruction set.
- Hardwired control typical of RISC.

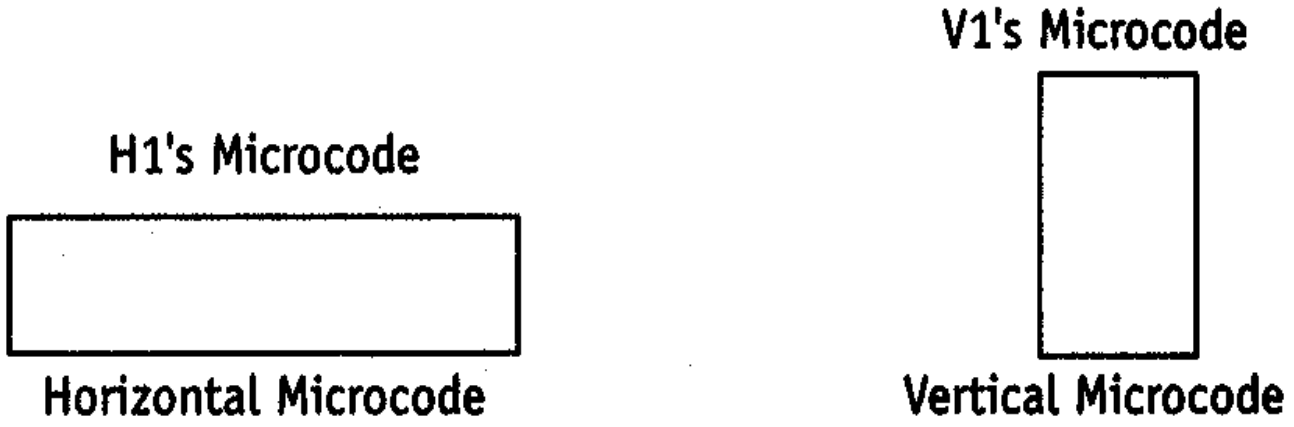
Hardwired control

FIGURE 6.19



Horizontal versus vertical microcode

FIGURE 6.20



Horizontal Microinstruction: 32 bits

Vertical microinstruction: 19 bits

sim supports both horizontal and vertical microcode. H1 is the horizontally microcode machine; V1 is the vertically microcoded machine.

To run **sim** with the built-in vertical microcode use the **/ver** command line argument.

```
C:\H1>sim a /ver
```

sim responds with

```
---- [T7] 0: ld   /0 018/
```

Now enter **enable**. **sim** responds with

```
Microlevel enabled
---- [T7] 0: ld   /0 018/
```

Next, set the microlevel display mode by entering **m**. **sim** responds with

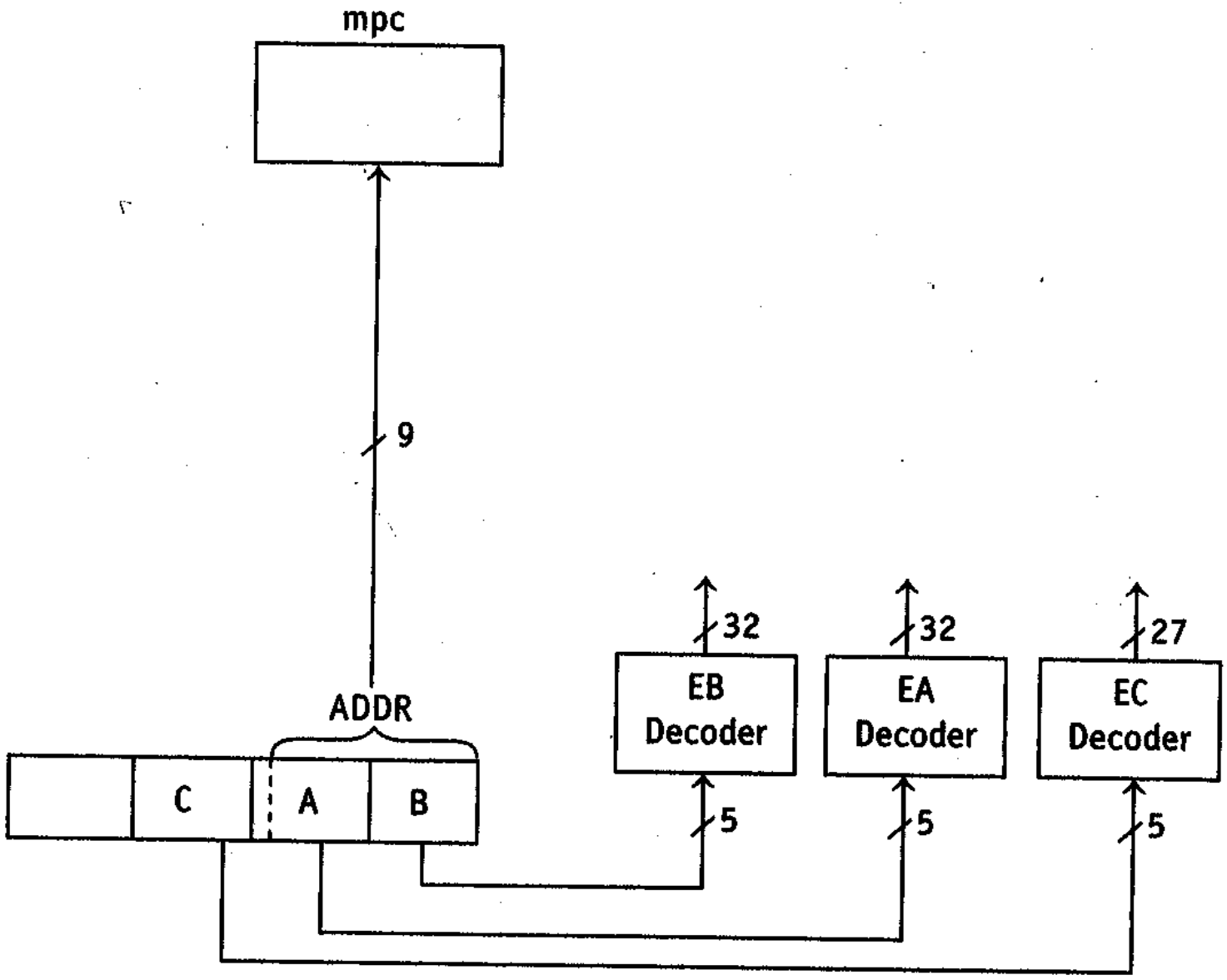
```
Microlevel display mode
---- [T1] 0: mar pc/
```

Observe that the debugger's prompt includes `mar pc`—the first vertical microinstruction in the standard vertical microcode. To see more vertical instructions, enter **u0** to unassemble from location 0. **sim** responds with

```
  0: mar     pc                          / 40006
  1: rd                                  / 68000
  2: move    ir       mdr                / 2A4A0
  3: add     pc       pc       1         / 018C1
---- [T1] 0: mar pc/
```

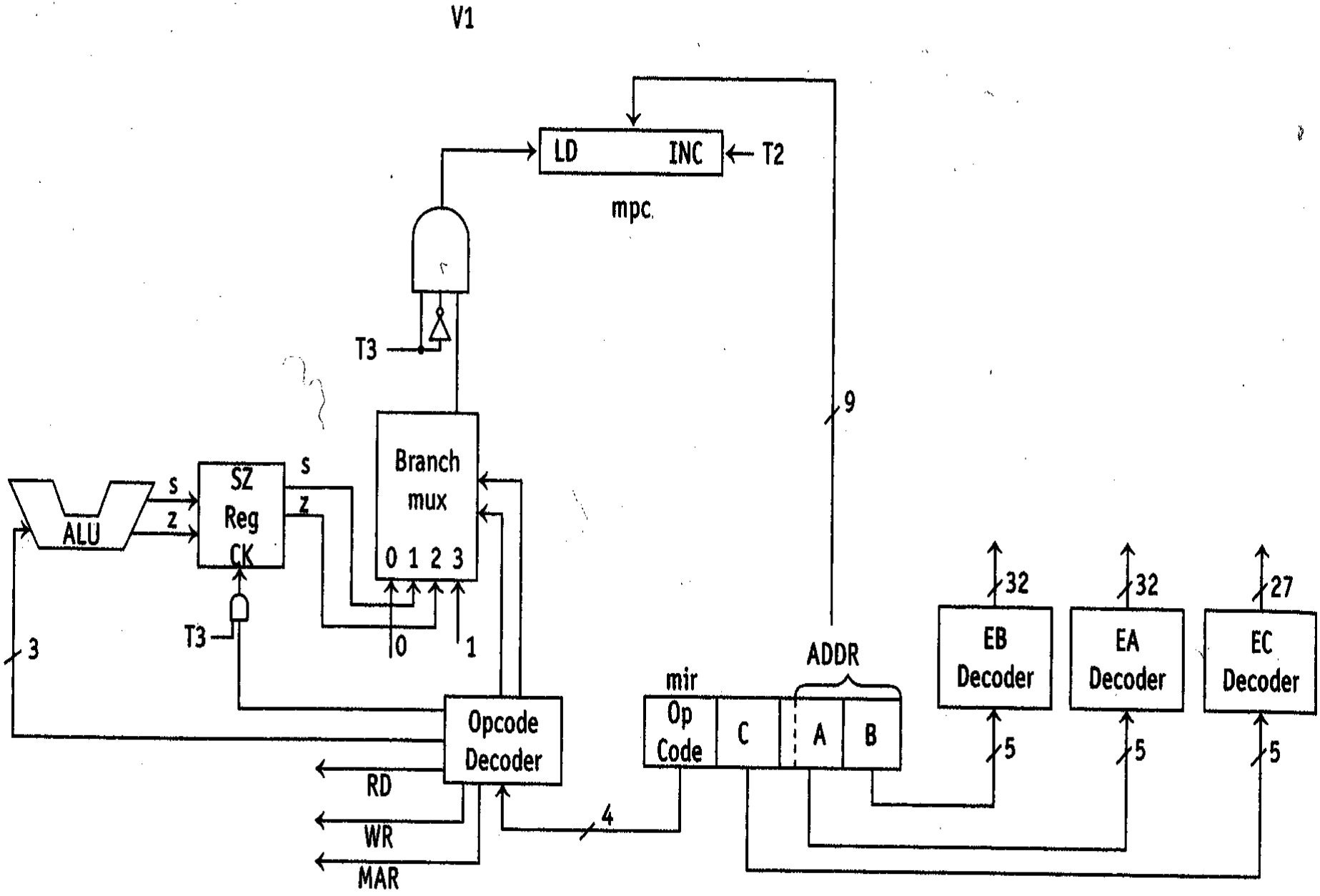
In V1, ADDR and the A/B fields overlap—this reduces the size of a microinstruction.

FIGURE 6.21 Address Field in Vertical Microinstruction



V1 uses additional decoding to reduce the size of microinstructions—an *opcode decoder*.

FIGURE 6.22


V1
LD
INC
T2
mpc
T3
9
Branch
mux
0 1 2 3
SZ
Reg
CK
s
z
ALU
T3
3
0
1
ADDR
mir
Op
Code
C
A
B
Opcode
Decoder
RD
WR
MAR
4
EB
Decoder
EA
Decoder
EC
Decoder
32
32
27
5
5
5

Opcode decoder

- 4-bit input
- 7 bits out: MAR, WR, RD, ALU function, SZ enable
- Reduces microinstruction size by 3 bits.

V1 cannot perform a computation and branch in one microinstruction. So the ALU output during a computation must be saved (in the SZ register) for a subsequent branch instruction.

See the next slide.

FIGURE 6.22

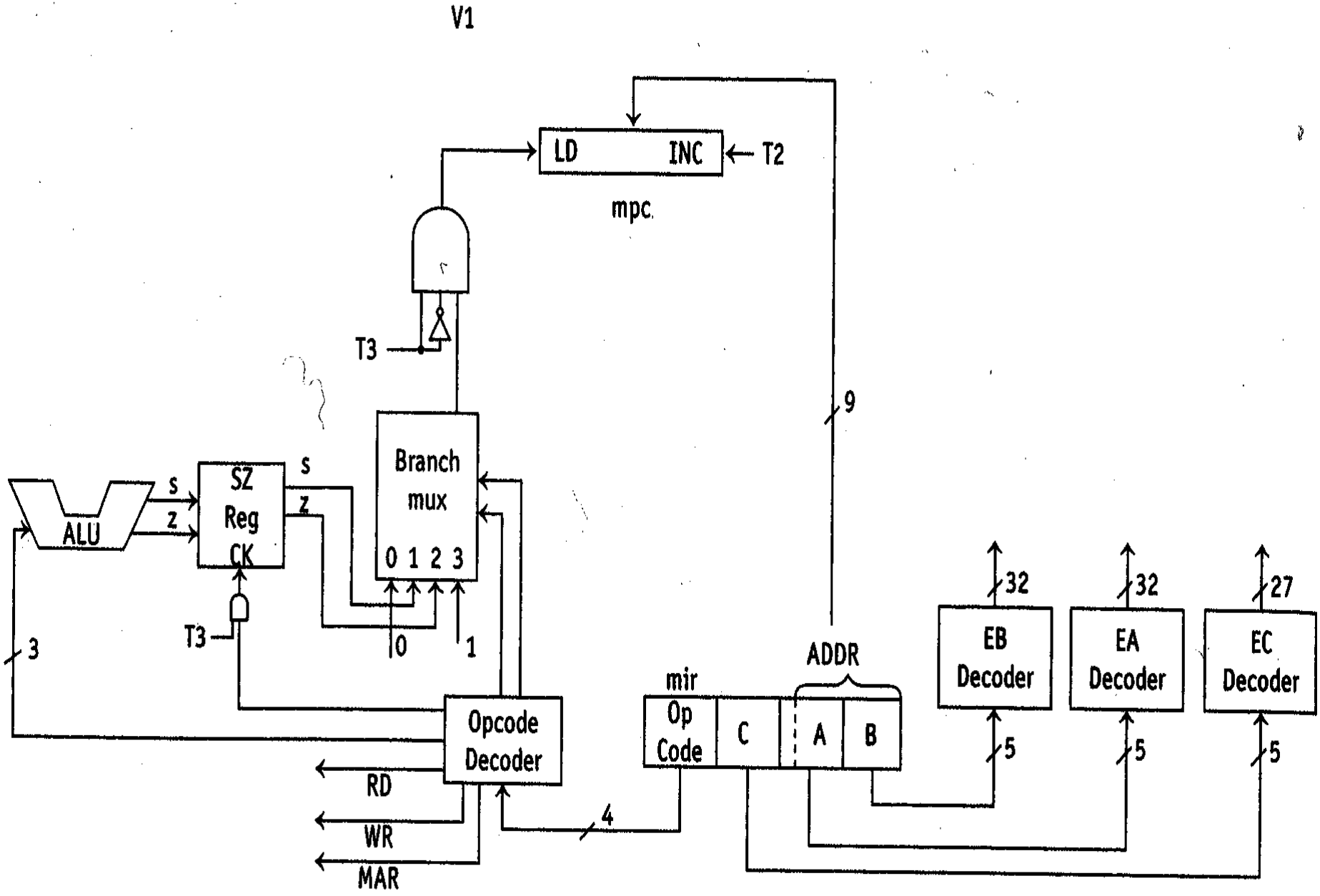

V1
LD
INC
T2
mpc
T3
9
Branch
mux
0 1 2 3
SZ
Reg
CK
s
z
ALU
T3
3
0
1
ADDR
mir
Op
Code
C
A
B
Opcode
Decoder
RD
WR
MAR
4
EB
Decoder
EA
Decoder
EC
Decoder
32
32
27
5
5
5

FIGURE 6.23

Opcode	(hex)	Assembly form	Name	Description
0	add	rc ra rb	Add	rc = ra + rb; set sz;
1	sub	rc ra rb	Subtract	rc = ra - rb; set sz;
2	mult	rc ra rb	Multiply	rc = ra * rb; set sz;
3	and	rc ra rb	Bitwise AND	rc = ra & rb; set sz;
4	flip	rc ra	Flip bits	rc = ~ra; set sz;
5	move	rc ra	Move register	rc = ra; set sz;
6	left	rc ra	Left shift	rc = left(ra); set sz;
7	right	rc ra	Right shift	rc = right(ra); set sz;
8	mar	rb	Load mar	mar = rb;
9	sz	ra	Set sz	set sz (with ra);
A	ba	addr	Branch always	mpc = addr;
B	bn	addr	Branch if neg	if (s) mpc = addr;
C	bz	addr	Branch if zero	if (z) mpc = addr;
D	rd	Read	mdr = mem[mar];	
E	wr	Write	mem[mar] = mdr;	

The basic instruction set requires more microinstructions in vertical microcode than in horizontal microcode (because one vertical instruction, in general, does less than one horizontal microinstruction.

FIGURE 6.24

```
1  / Basic Instruction Set Vertical Microcode       b.vas
2
3  / The label indicates the opcode decoded up to that point.
4  / For example, control reaches the label L011 when opcode
5  / bits 011 have been decoded.
6
7  /*********************************************************/
8  /       Fetch instruction and increment pc register      /
9  /*********************************************************/
10 fetch:        mar  pc
11               rd                        / fetch mach inst
12               move ir mdr               / move inst to ir
13               add  pc pc 1              / increment pc
14 /*********************************************************/
15 /              Decode opcode                             /
16 /*********************************************************/
17               move   dc ir
18               bn     L1                 / test 1st bit
19 L0:           left   dc dc
20               bn     L01                / test 2nd bit
21 L00:          left   dc dc
22               bn     L001               / test 3rd bit
23               ba     L000
24 L1:           left   dc dc
25               bn     L11                / test 2nd bit
26 L10:          left   dc dc
27               bn     L101               / test 3rd bit
28               ba     L100
29 L01:          left   dc dc
30               bn     L011               / test 3rd bit
31               ba     L010
32 L11:          left   dc dc
33               bn     L111               / test 3rd bit
34               ba     L110
35 /*********************************************************/
36 /              Microcode for each instruction            /
37 /*********************************************************/
38 L000:          /--------------------  LD  ----------------
39               mar    ir                 / get address
40               rd                        / fetch operand
41               move   ac mdr             / move operand to ac
42               ba     fetch
```

(continued)

FIGURE 6.24 (continued)

```
43 L001:    /--------------------  ST  ------------------
44          move    mdr ac          / prepare for write
45          mar     ir
46          wr                      / write
47          ba      fetch           / branch
48 L010:    /--------------------- SHLL -----------------
49          and     f ir zmask      / get z-field
50          bz      fetch           / branch if 0
51 lloop:   left    ac ac           / shift ac left
52          sub     f f 1           / decrement count
53          bz      fetch           / branch if 0
54          ba      lloop           / branch
55 L011:    /--------------------- SHRL -----------------
56          and     f ir zmask      / get z-field
57          bz      fetch           / branch if 0
58 rloop:   right   ac ac           / shift ac right
59          sub     f f 1           / decrement count
60          bz      fetch           / branch if 0
61          ba      rloop           / branch
62 L100:    /--------------------- ADDC -----------------
63          and     f ir xmask      / get x-field
64          add     ac ac f         / add to ac
65          ba      fetch           / branch
66 L101:    /--------------------- FLIP -----------------
67          flip    ac ac           / bitwise compl ac
68          ba      fetch           / branch
69 L110:    /--------------------- MULT -----------------
70          mar     ir              / get address
71          rd                      / fetch operand
72          mult    ac ac mdr       / multiply ac by operand
73          ba      fetch           / branch
74 L111:    /--------------------  JN  ------------------
75          move    0 ac            / get ac
76          bn      dojn            / branch is negative
77          ba      fetch           / branch
78 dojn:    and     pc ir xmask     / put x-field in pc
79          ba      fetch           / branch
```

Using new vertical microcode b.vas with configuration file b.cfg

- Assemble microcode: **vas b**
  (translates b.vas to b.ver). vas responds with
  Reading configuration file b.cfg
- Assemble test program: **mas bprog**
  (translates bprog.mas to bprog.mac). mas responds with
  Reading configuration file b.cfg
- Run bprog on sim: **sim bprog /mb.ver**
  sim responds with
  Reading configuration file b.cfg
  Reading microcode file b.ver

FIGURE 6.25

```
Starting session. Enter h or ? for help.
---- [T7] 0: ld   /0 00A/ t2
  0: ld   /0 00A/ ac=0000/FFFF
  1: flip /A 000/ ac=FFFF/0000
---- [T2] 2: addc /8 003/ enable
Microlevel enabled
---- [T2] 2: addc /8 003/ m
Microlevel display mode
---- [T1] 0: mar pc/ t2
  0: mar pc
     mar=001/002
  1: rd
     Rd from m[002] mdr=A000/8003  2: addc /8 003/
---- [t2] 2: move ir mdr/ u0
  0: mar     pc                          / 40006
  1: rd                                  / 68000
  2: move    ir       mdr                / 2A4A0
  3: add     pc       pc       1         / 018C1
---- [t2] 2: move ir mdr/ d0
  0: 40006  68000  2A4A0  018C1  2A920  5800B  32940  58010
  8: 32940  5801A  50016  32940  58013  32940  5802D  5002A
 10: 32940  58024  5001E  32940  58033  5002F  40009  68000
 18: 2A0A0  50000  29500  40009  70000  50000  1BD24  60000
---- [t2] 2: move ir mdr/ R*
     pc    = 0002     ac    = 0000
---- [t2] 2: move ir mdr/ n
No display mode
---- [T7] g
6
Machine inst count =      A (hex) =       10 (dec)
Micro   inst count =     7E (hex) =      126 (dec)
---- [T7] q
```

Comparing microcode size (total bits) of H1 and V1—V1's slightly smaller microcode size does not justify its slower execution.

H1 $36 \times 32 \text{ bits} = 1152 \text{ bits}$

But the total microcode bits for V1 is

V1 $56 \times 19 \text{ bits} = 1064 \text{ bits}$

8% reduction in size in total bits